



University of
Pittsburgh

Algorithms and Data Structures 2

CS 1501



Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
 - Homework 3: this Friday @ 11:59 pm
 - Lab 2: Tuesday 2/7 @ 11:59 pm
 - Assignment 1: Friday 2/17 @ 11:59 pm
- Please make your Piazza posts public as much as possible

Previous lecture

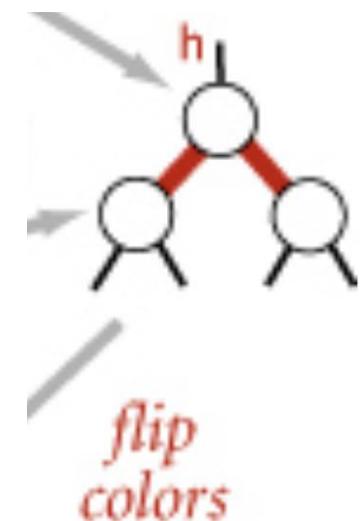
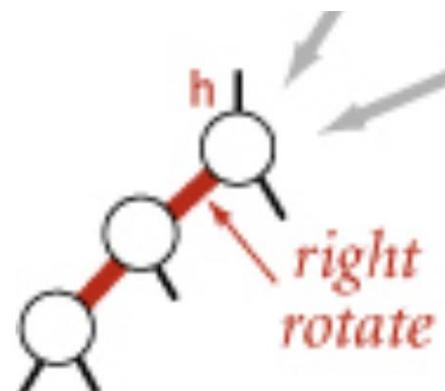
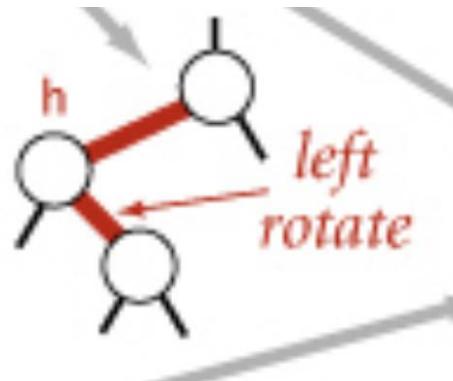
- Red-Black BST (self-balancing BST)
 - definition and basic operations
 - delete
 - runtime of operations
- Turning recursive tree traversals to iterative

Today ...

- Red-Black BST (self-balancing BST)
 - add
 - delete
 - runtime of operations
- Turning recursive tree traversals to iterative

Fixing Violations using Recursion

- Violations are checked and corrected as we climb back up the tree.
- The code for violation corrections is after the recursive calls.
- Violations are detected and corrected in the following order (h is the current node)



Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



A red circle containing the number 1.

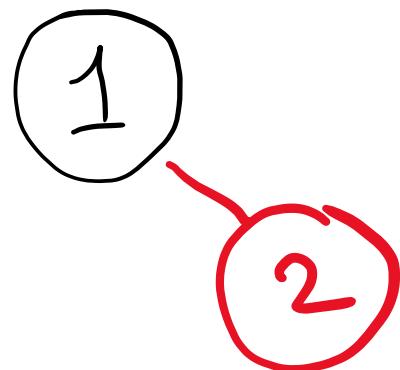
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7

1

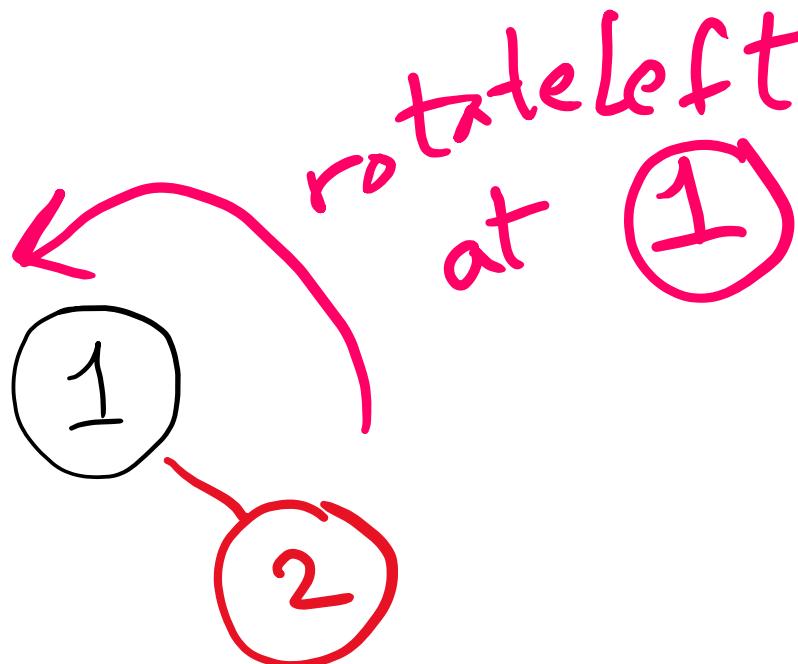
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



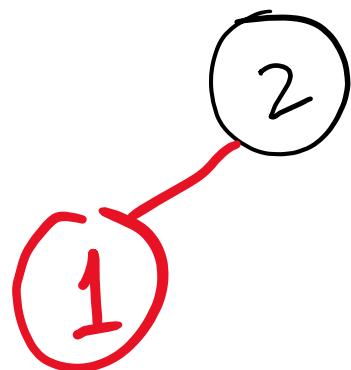
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



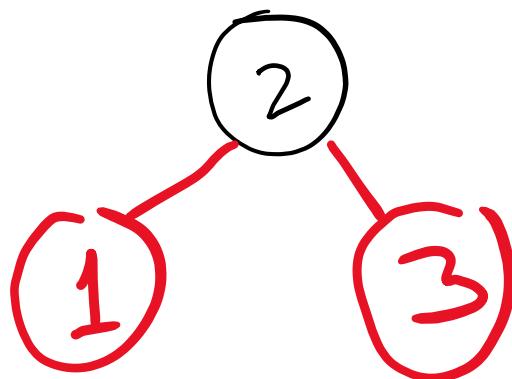
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



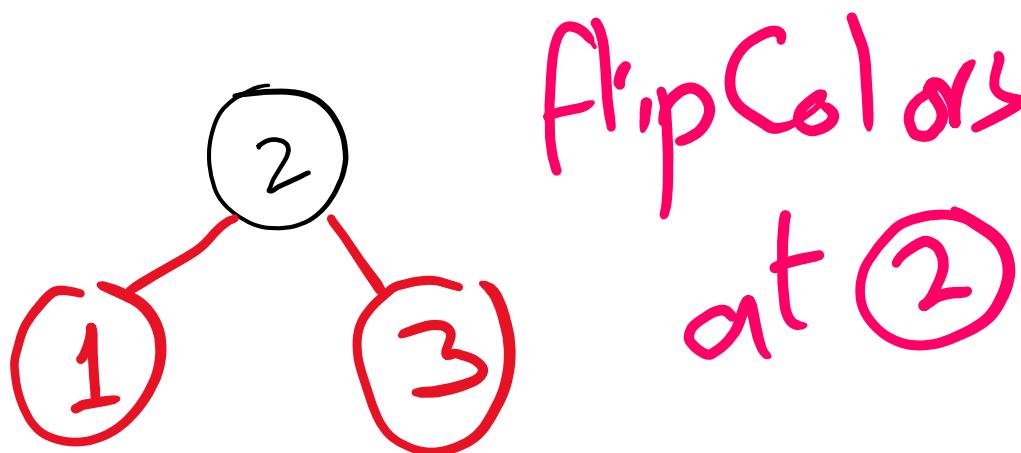
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



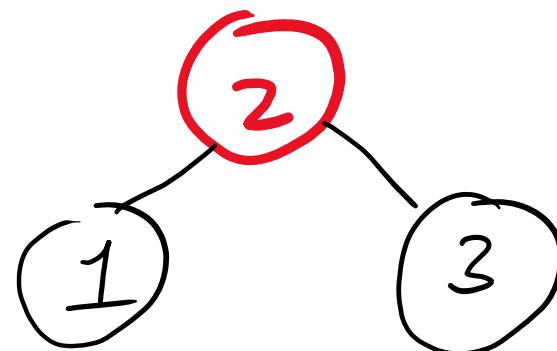
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



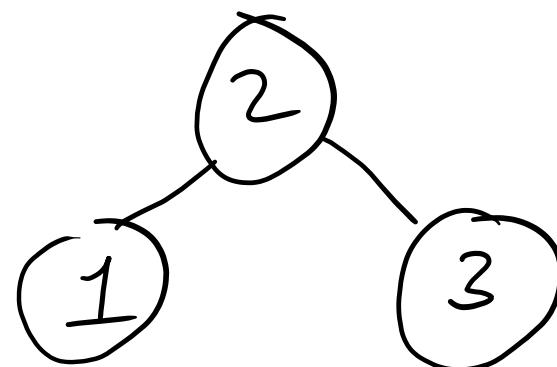
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



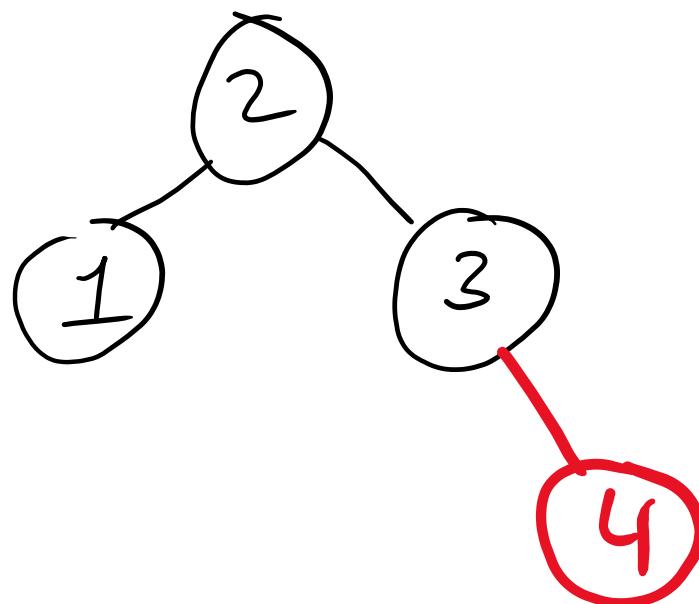
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



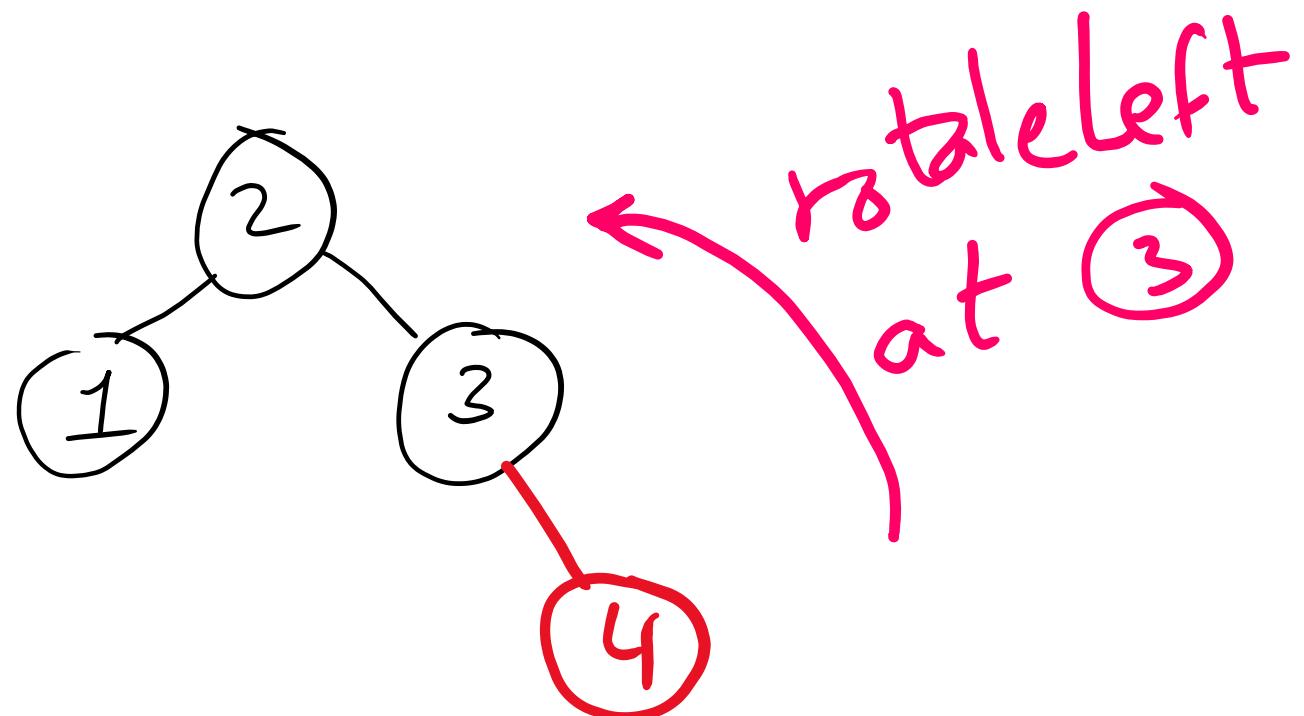
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



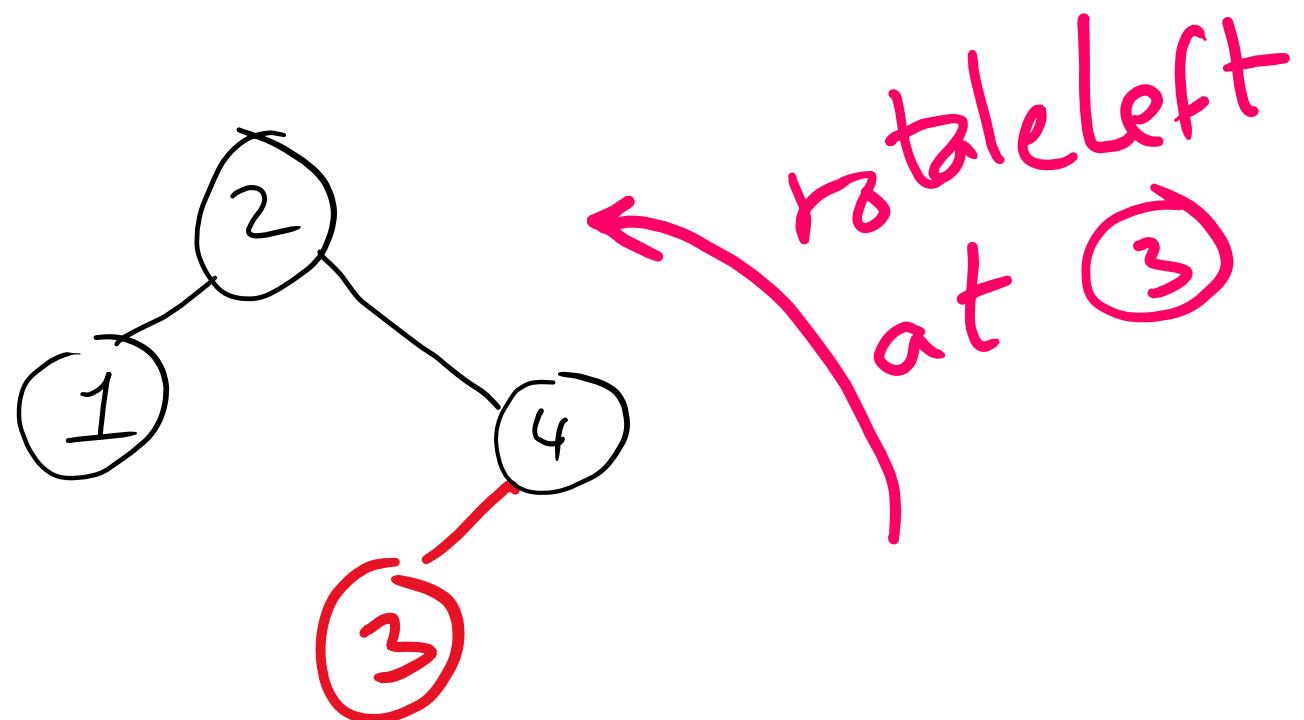
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



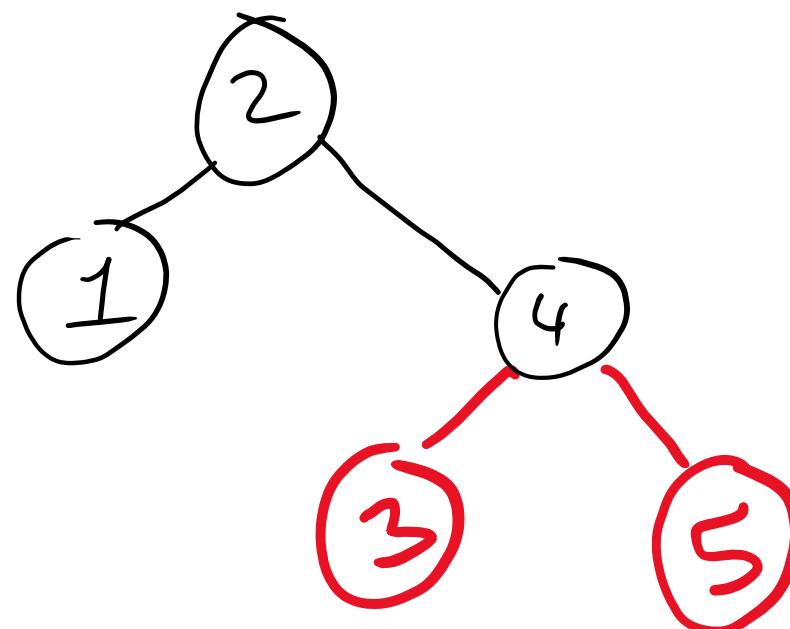
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



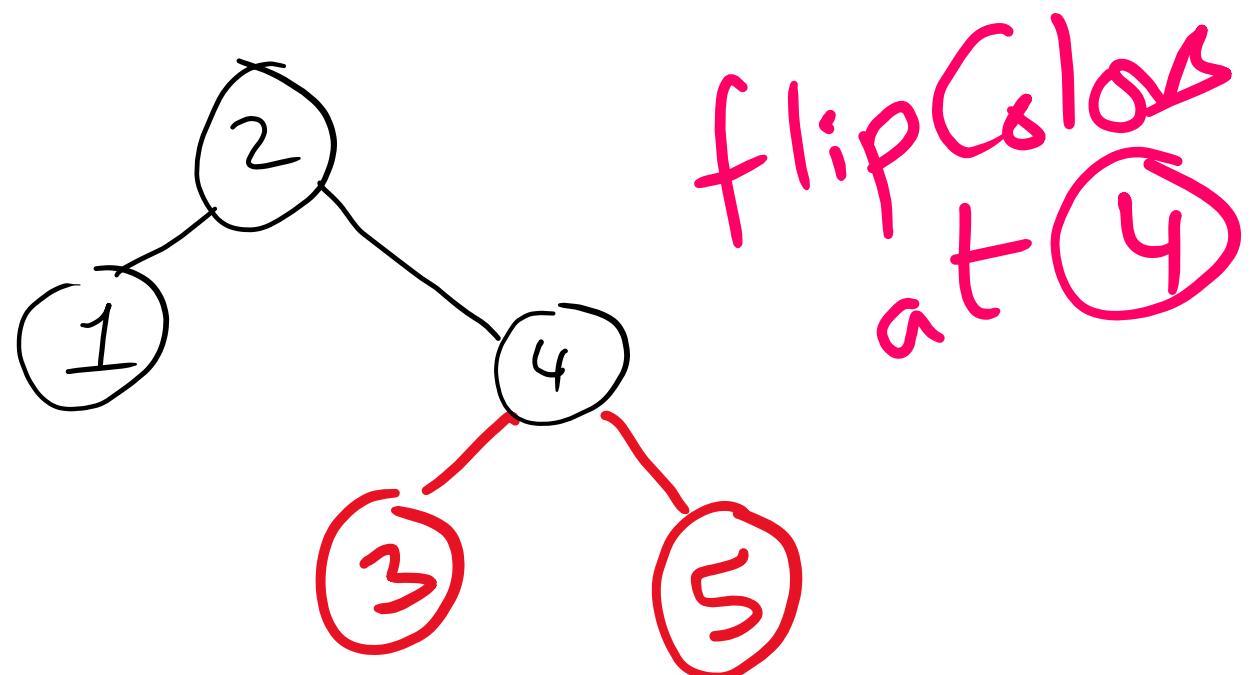
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



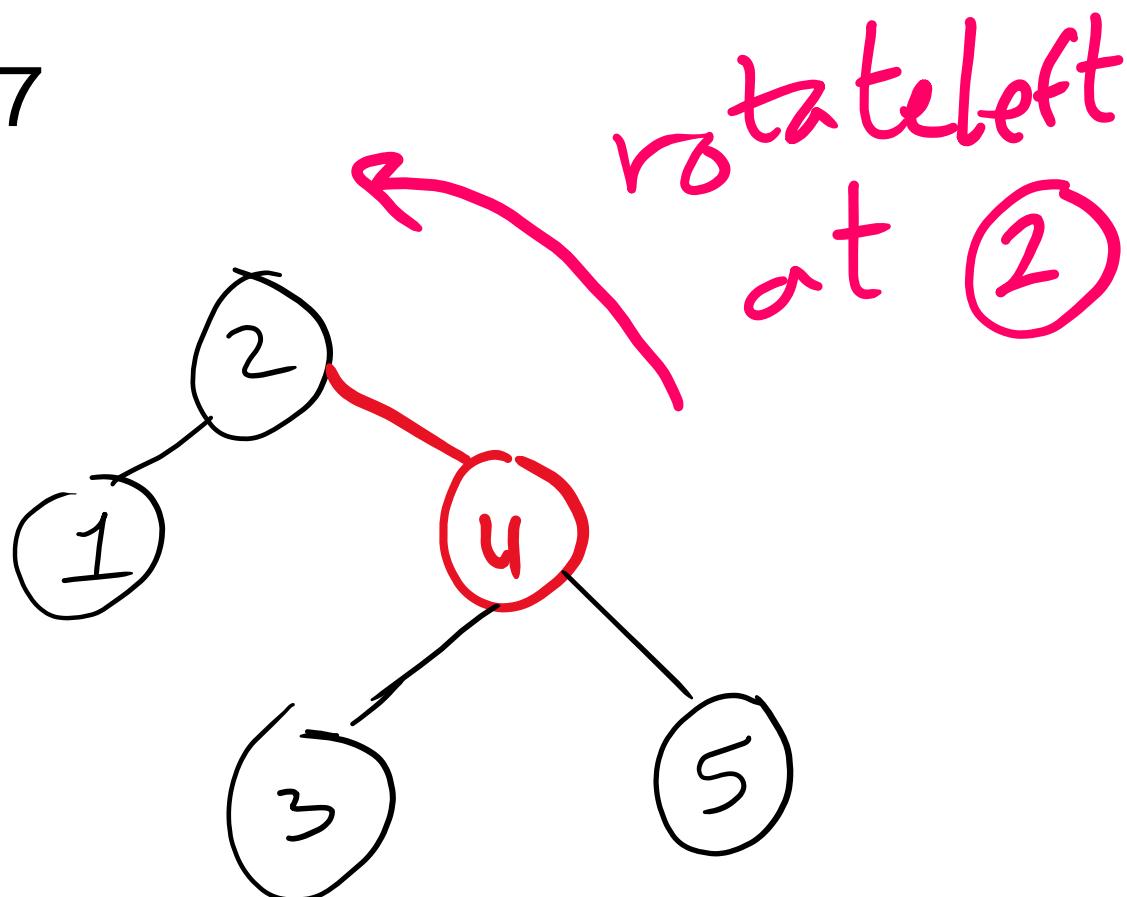
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



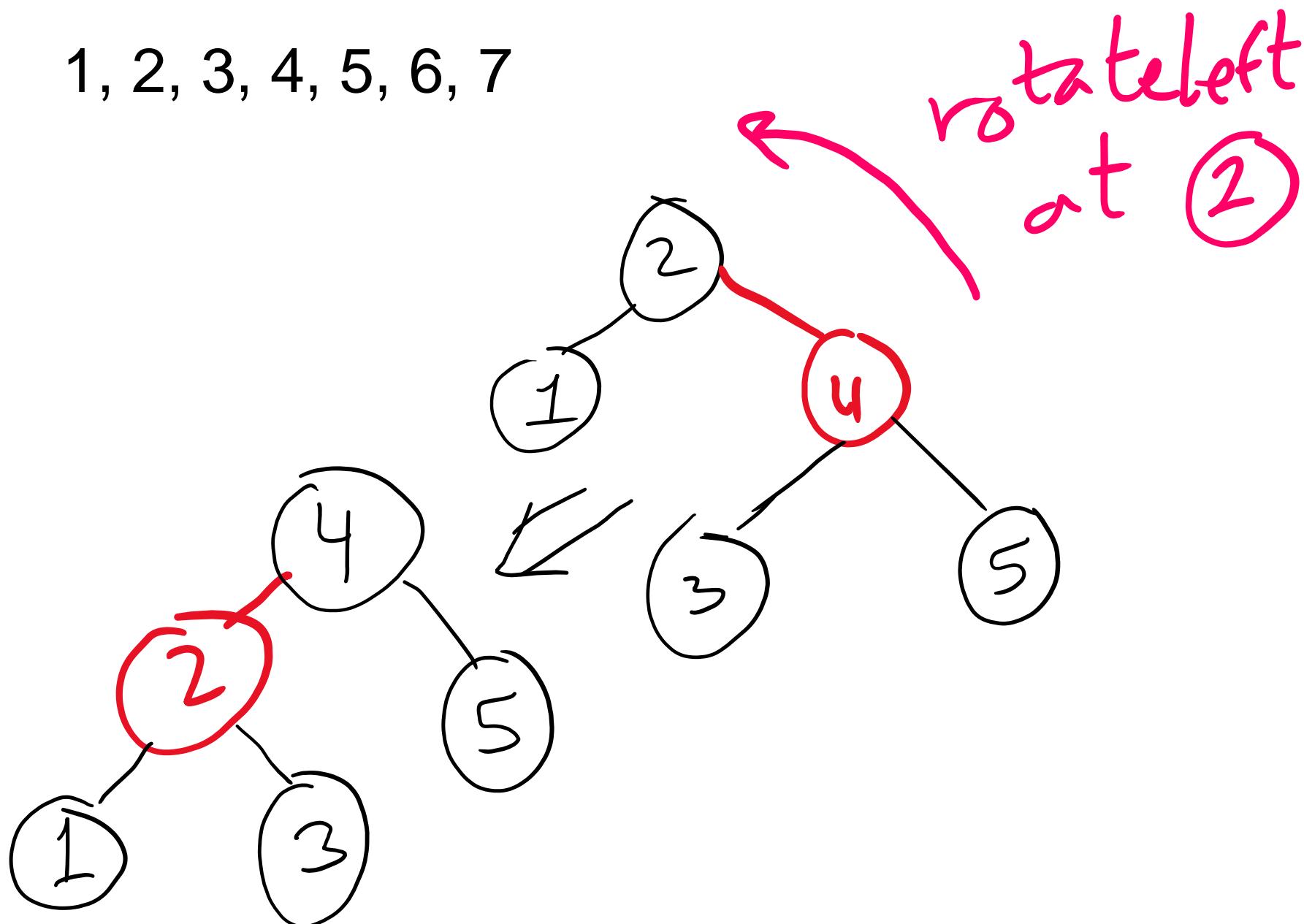
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



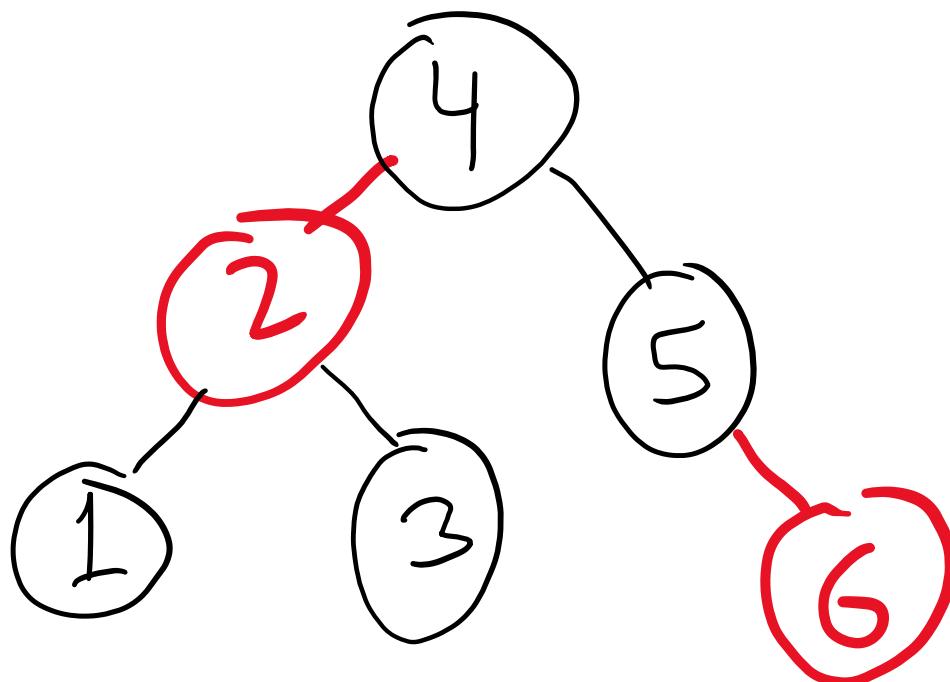
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



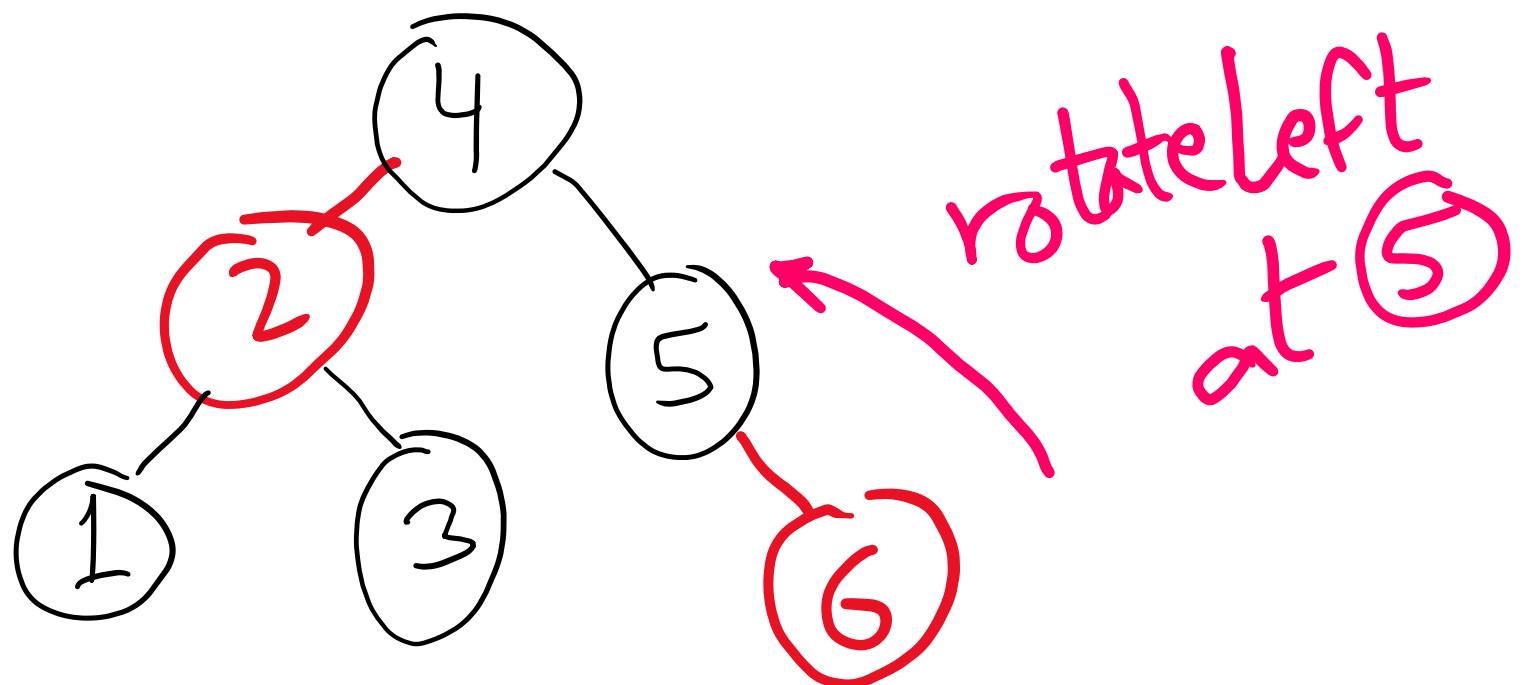
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



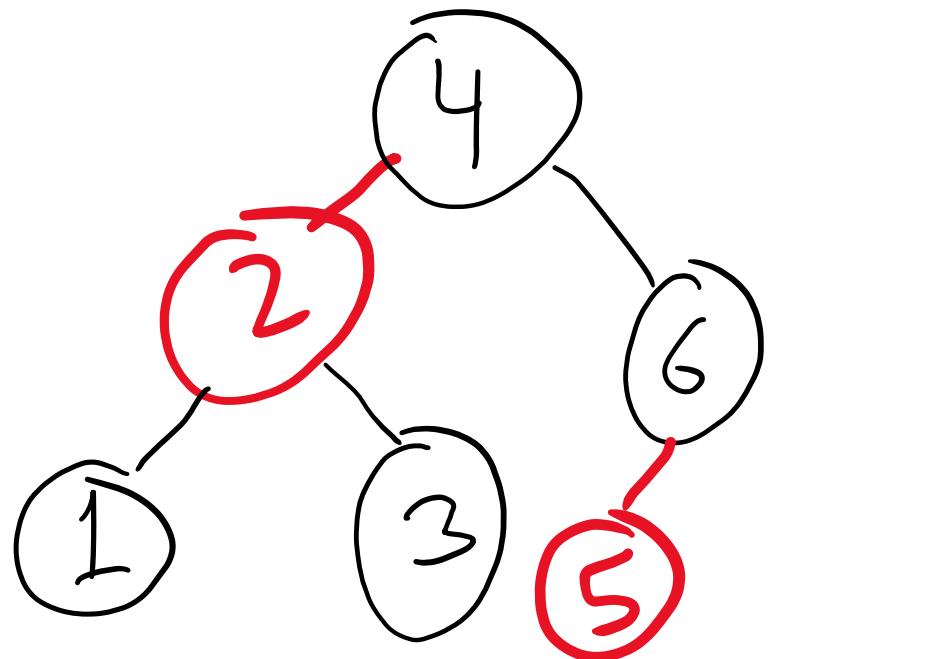
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



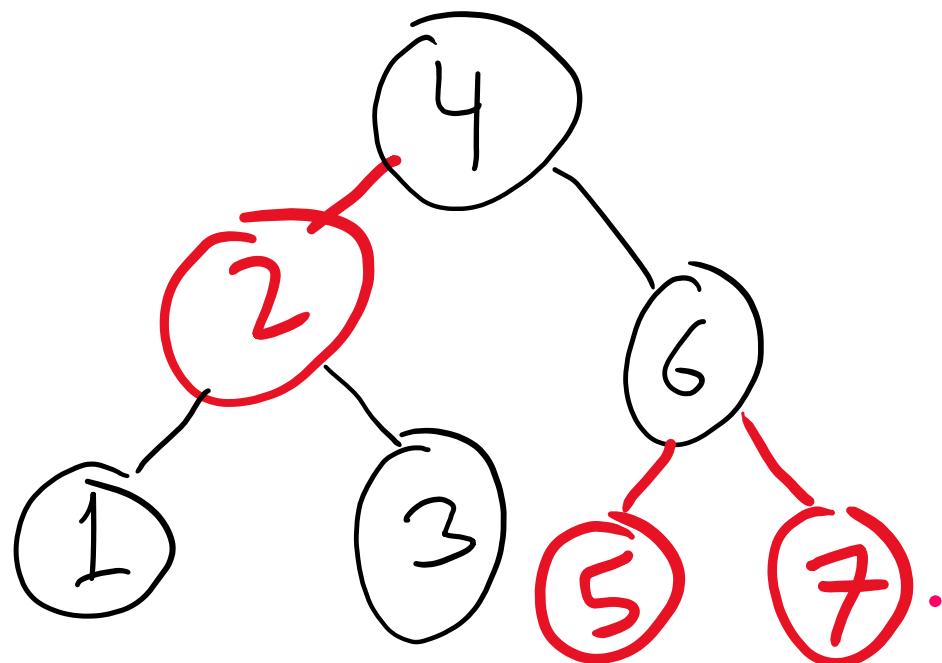
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



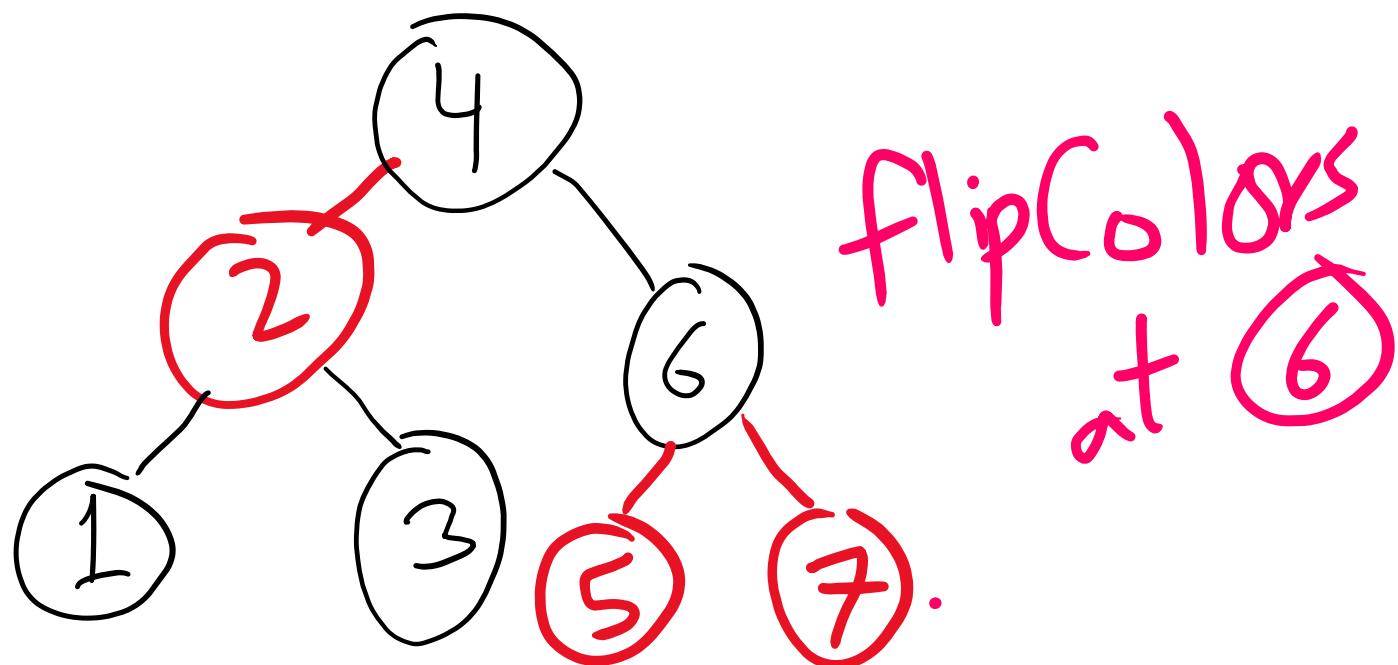
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



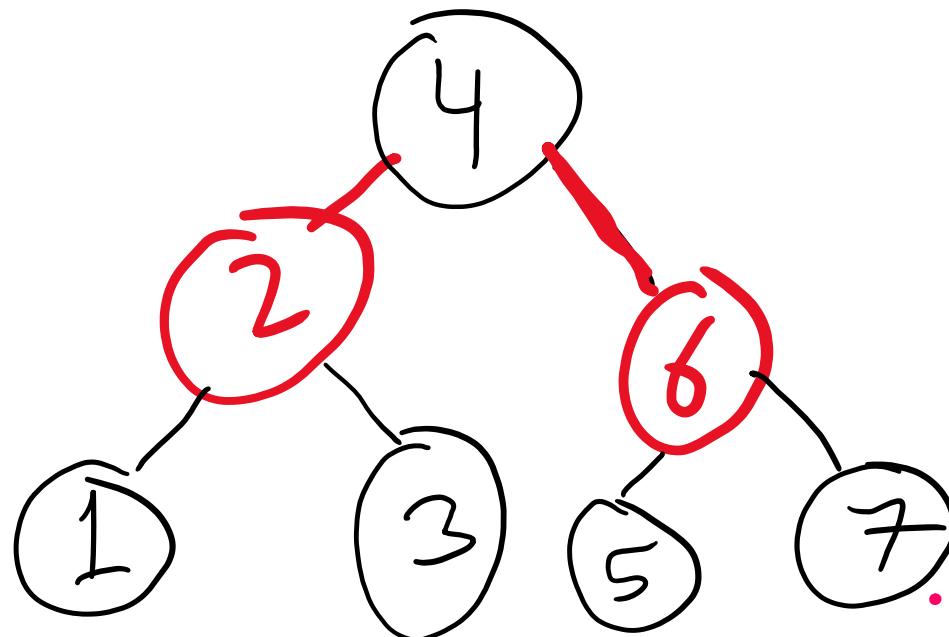
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



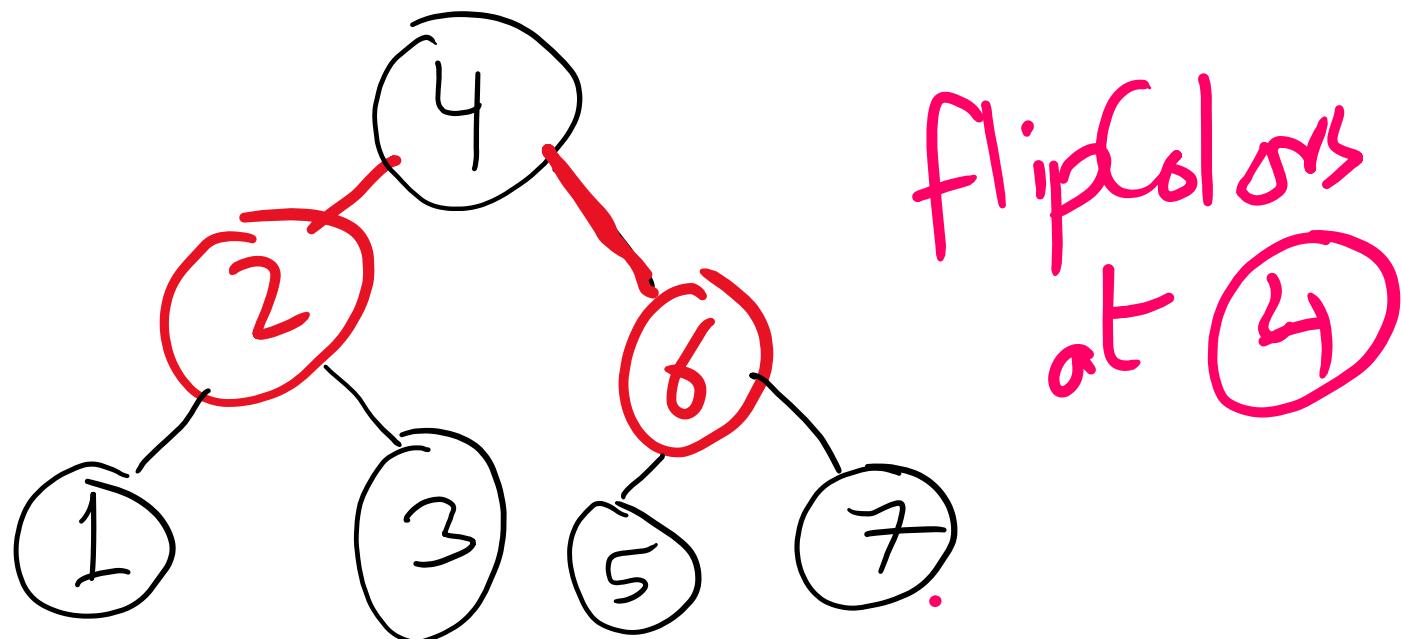
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



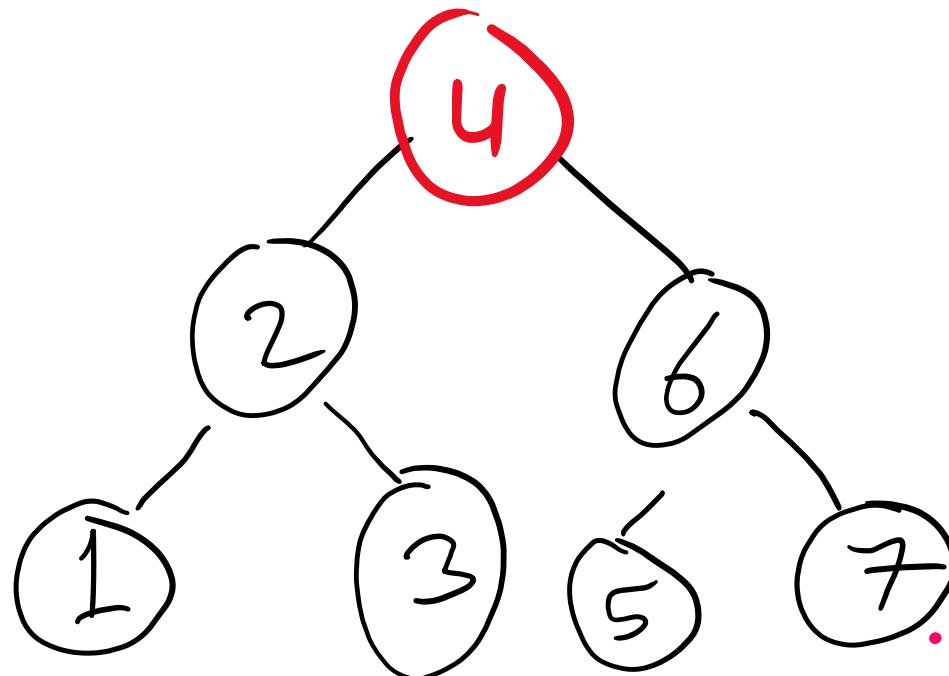
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



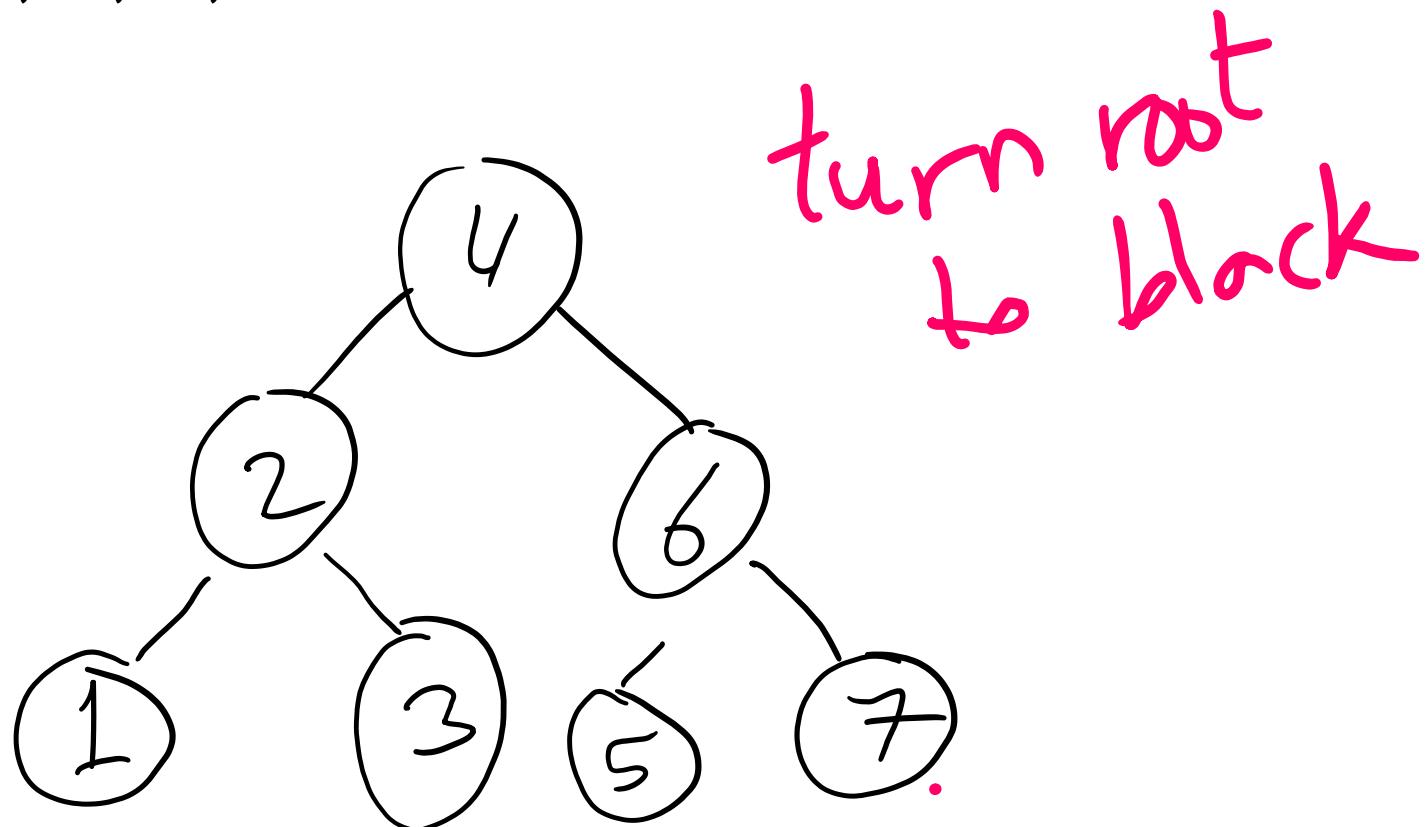
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



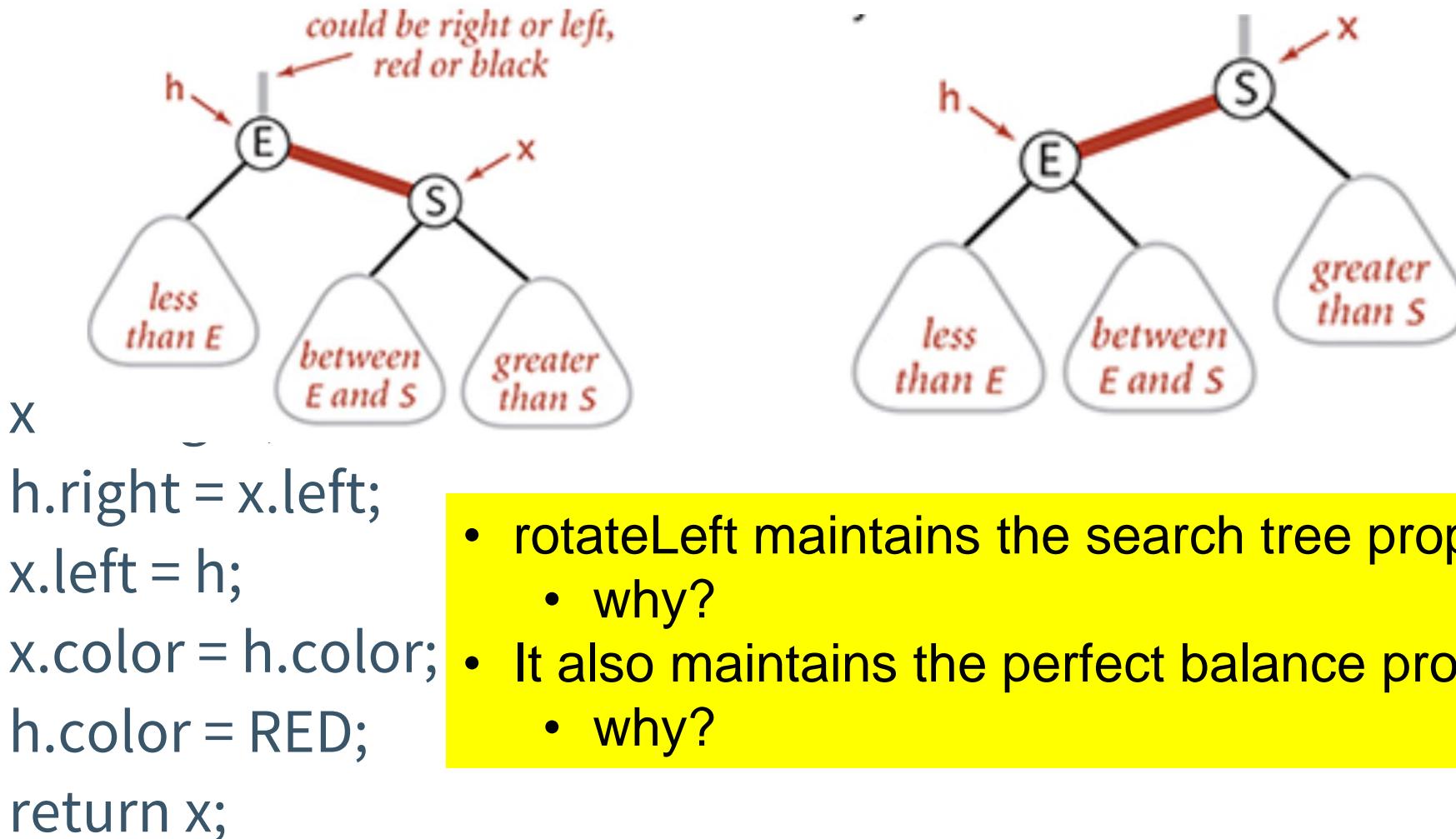
Adding to a RB-BST: Example

- Let's add the numbers:
- 1, 2, 3, 4, 5, 6, 7



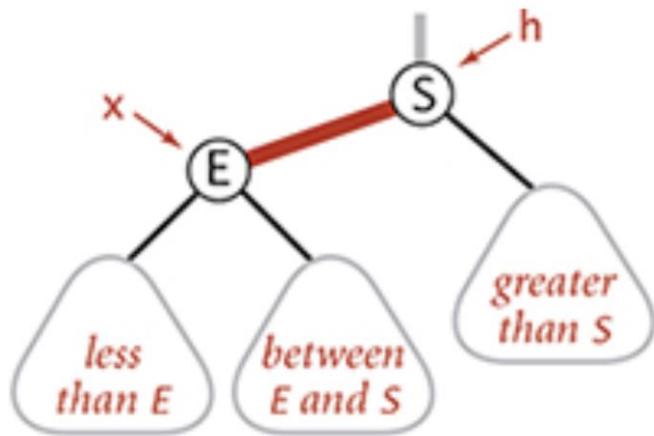
Left Rotate

- **BinaryNode<T> rotateLeft(BinaryNode<T> h)** performs left rotation at node h . The code emerges from contrasting the before and after pictures



Right Rotate

- **BinaryNode<T> rotatRight(BinaryNode<T> h)** performs right rotation at node h . The code emerges from contrasting the before and after pictures



$x = h.left;$

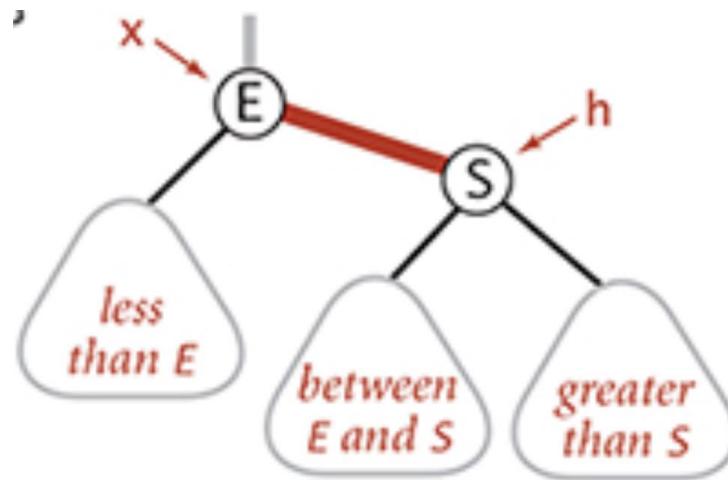
$h.left = x.right;$

$x.right = h;$

$x.color = h.color;$

$h.color = \text{RED};$

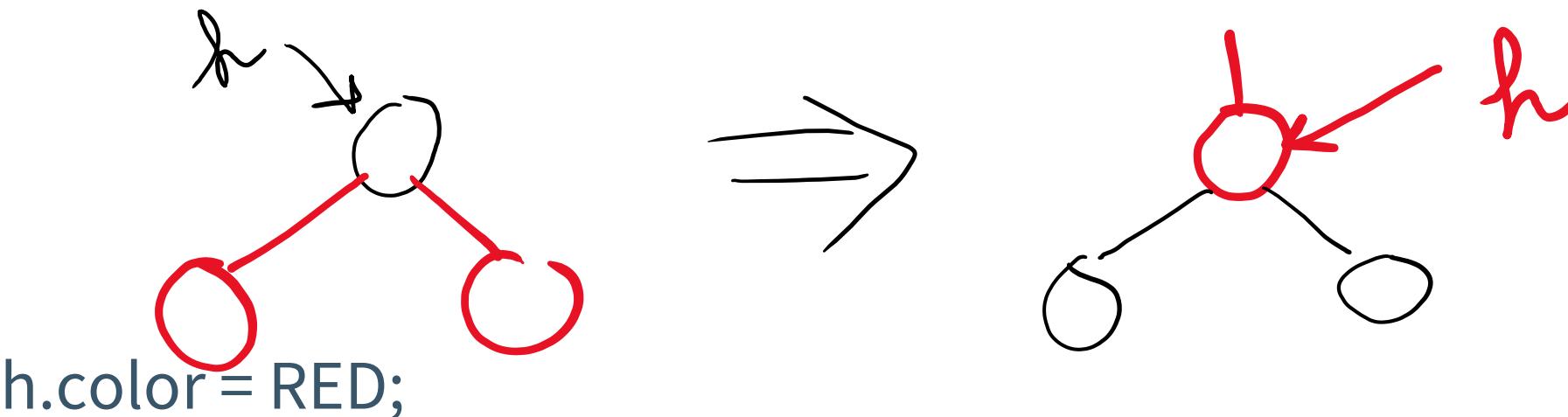
$\text{return } x;$



- `rotateRight` maintains the search tree property
 - why?
- It also maintains the perfect balance property
 - why?

Color Flip

- **BinaryNode<T> flipColors(BinaryNode<T> h)** performs color flip at node h . The code emerges from contrasting the before and after pictures



`h.color = RED;`

`h.right.color = BLACK;`

`h.left.color = BLACK;`

`return h;`

- `flipColors` maintains the search tree property
 - why?
- It also maintains the perfect balance property
 - why?

Deleting a node

- Make sure that we are not deleting a black node
 - as we go down the tree, make sure that the next node down is red
 - using a different set of operations
 - as we go back up the tree, correct any violations
 - same as we did while adding
 - if deleting a node with 2 children
 - replace with minimum of right subtree
 - delete minimum of right subtree
 - similar trick to delete in regular BST

Other BST operations

- Find successor and predecessor of an item
 - Lab 3
- Find all items within a specific range
 - Please check the keys methods in RedBlackBST.java inside the TreeADT folder in the code handouts
- Same code as regular BST!
- **worst-case runtime = Theta(log n)**

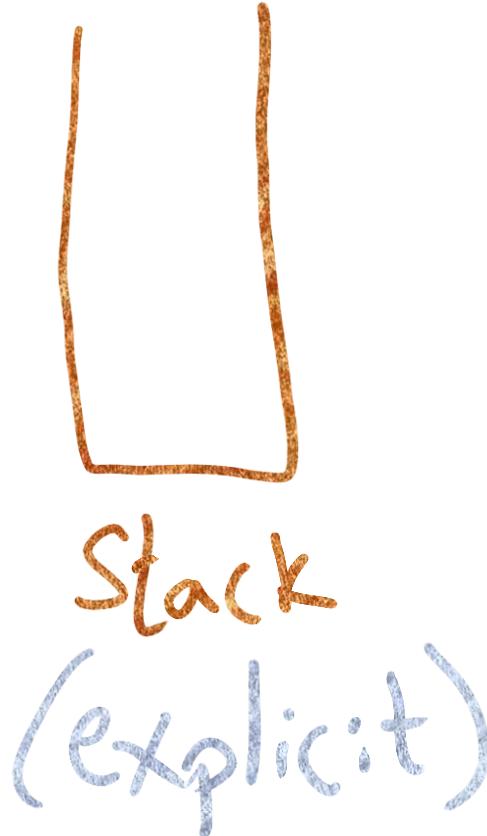
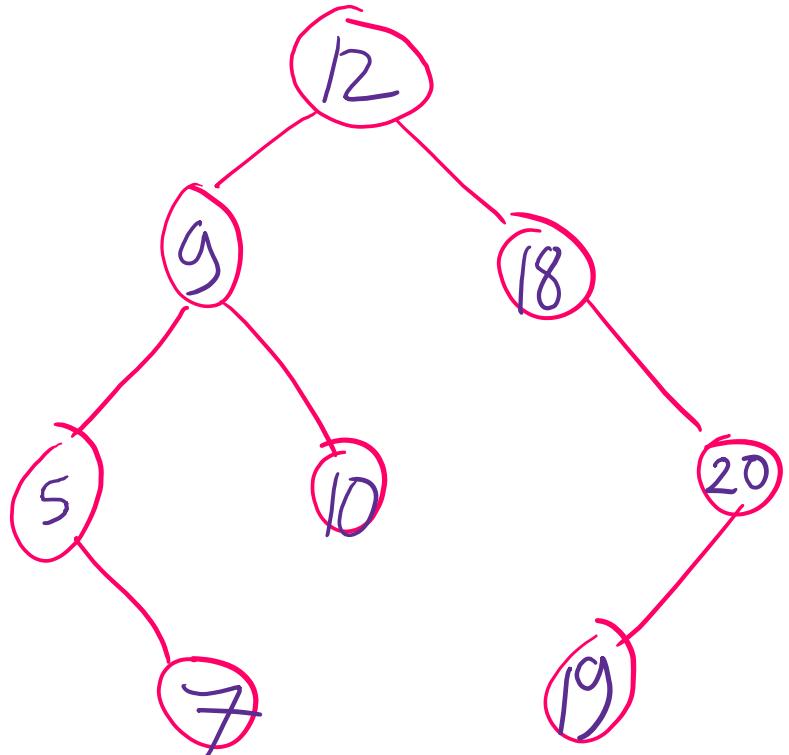
Symbol Table Implementations

	Unsorted Array	Sorted Array	Unsorted L.L.	Sorted L.L.	BST	RB BST
add	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
Search	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
		Binary Search		Binary Search		

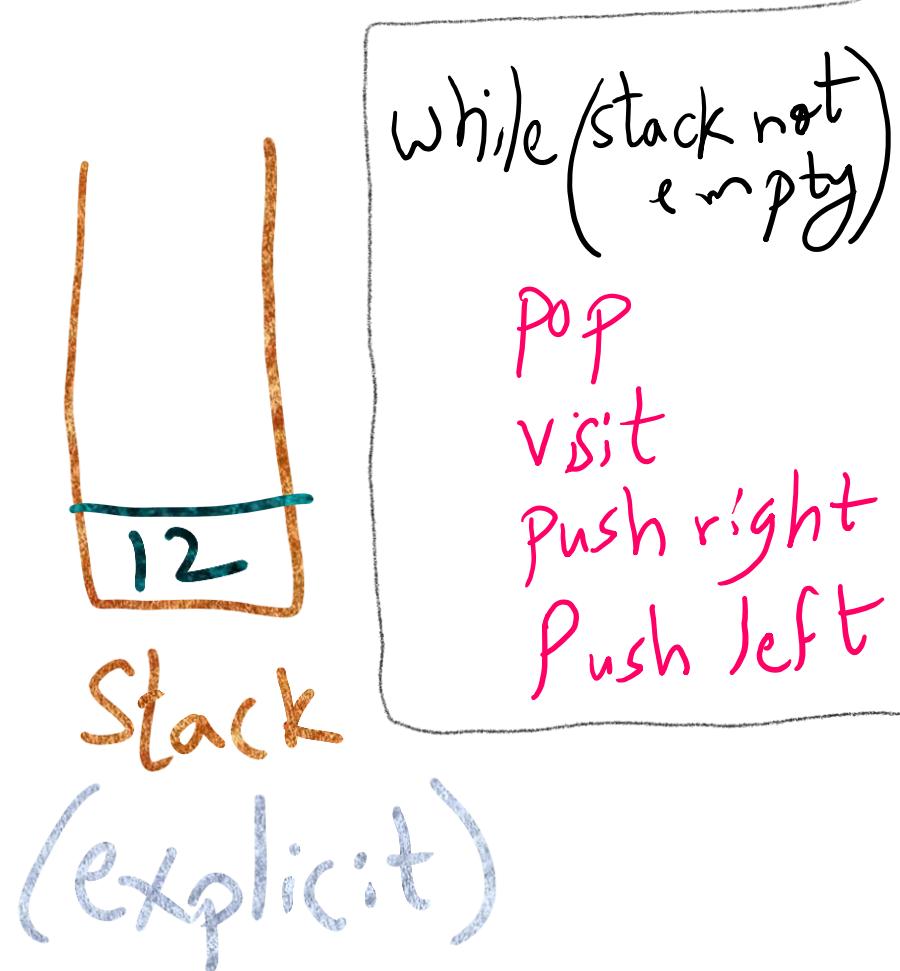
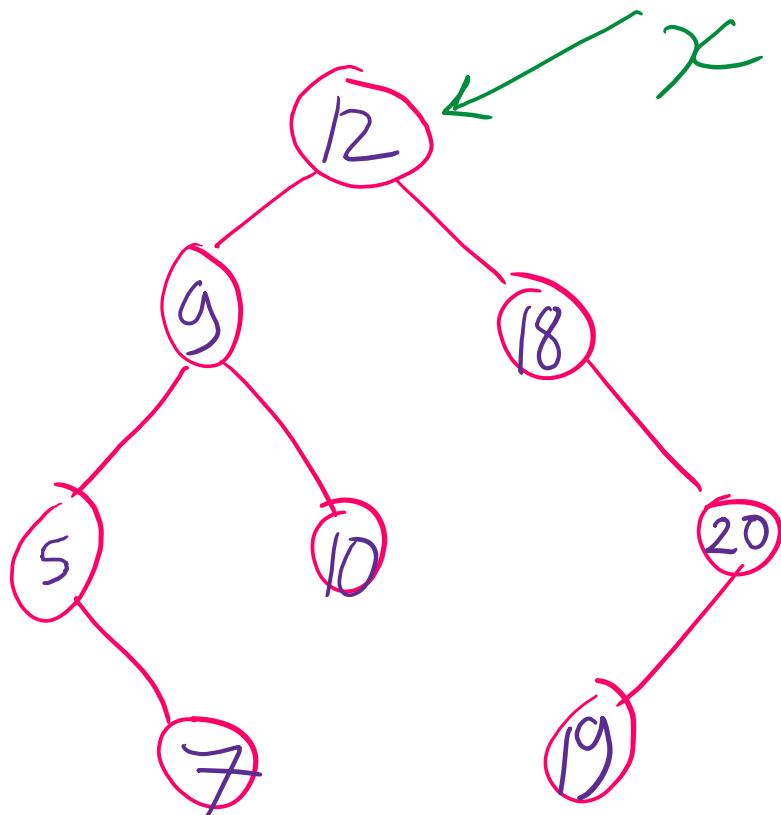
Why use iteration instead of recursion?

- Iteration is faster than recursion
 - No function call overhead (stack allocation)
- Especially useful for frequently used operations
 - Search is a good example of these operations

Iterative Preorder traversal

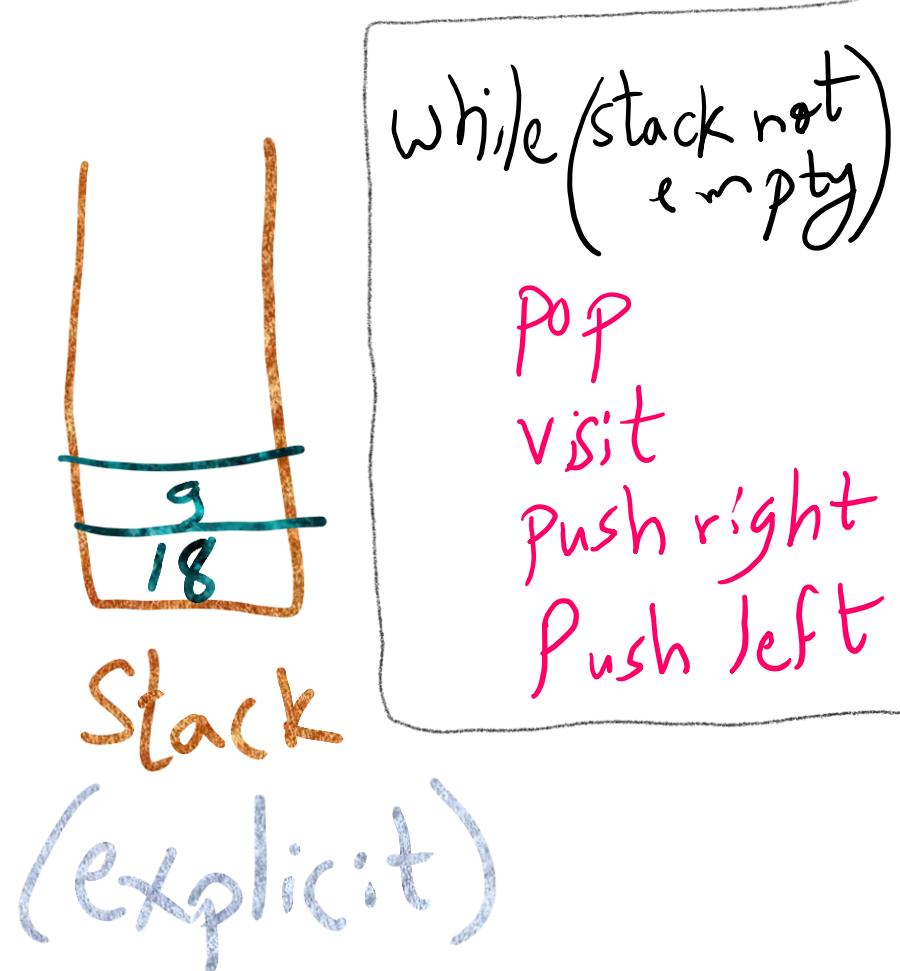
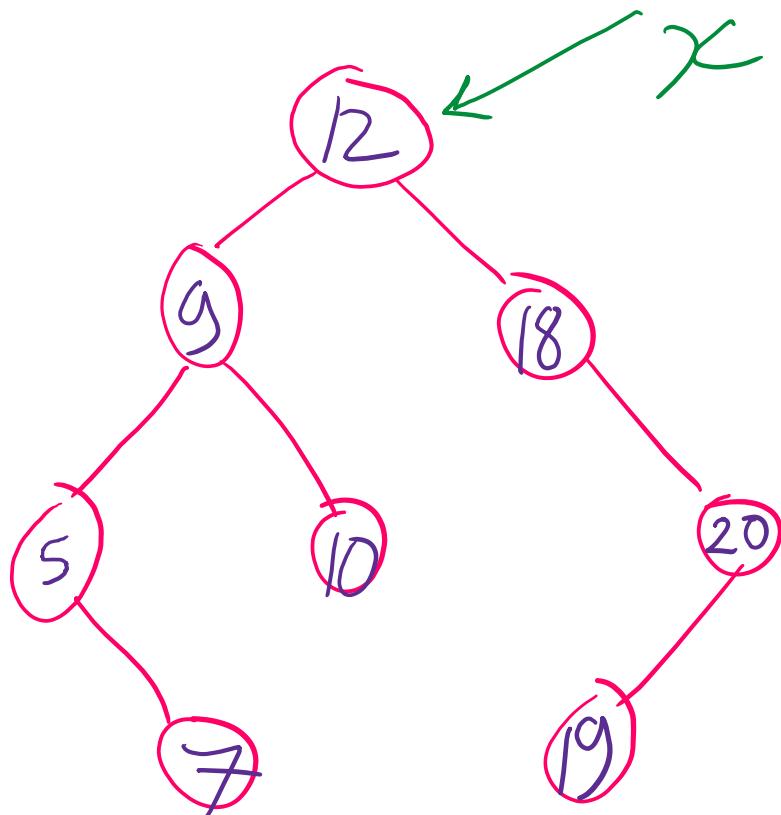


Iterative Preorder traversal



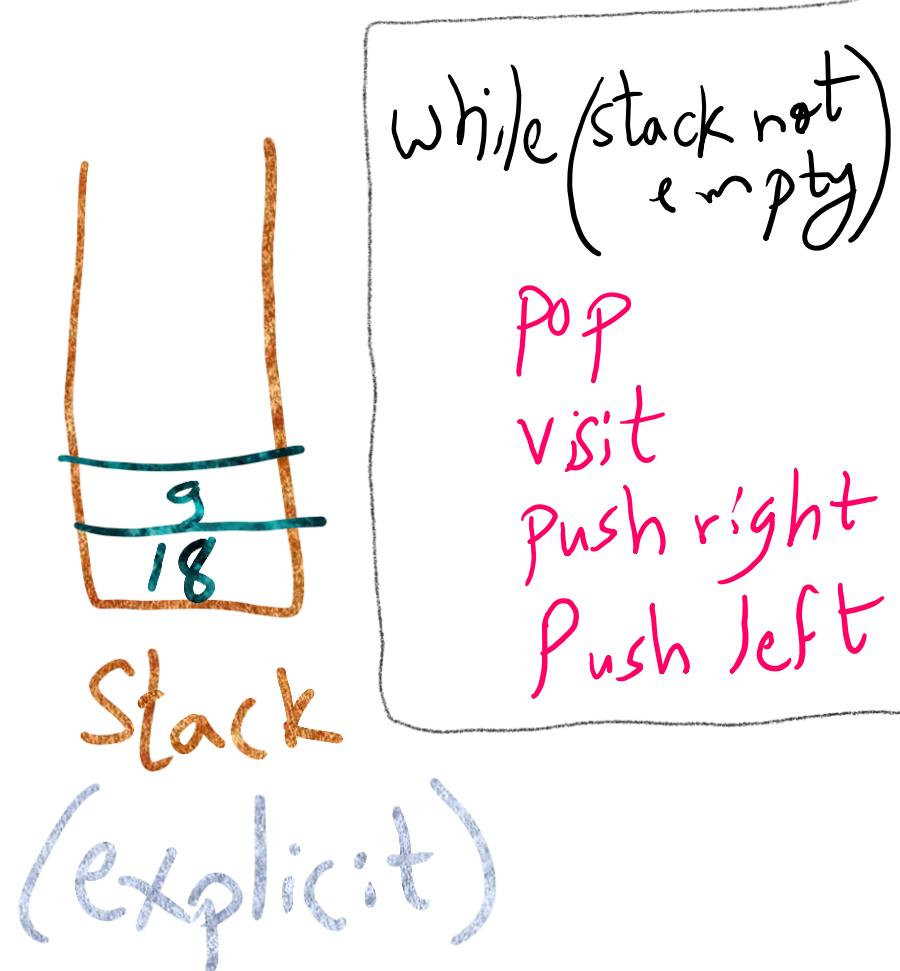
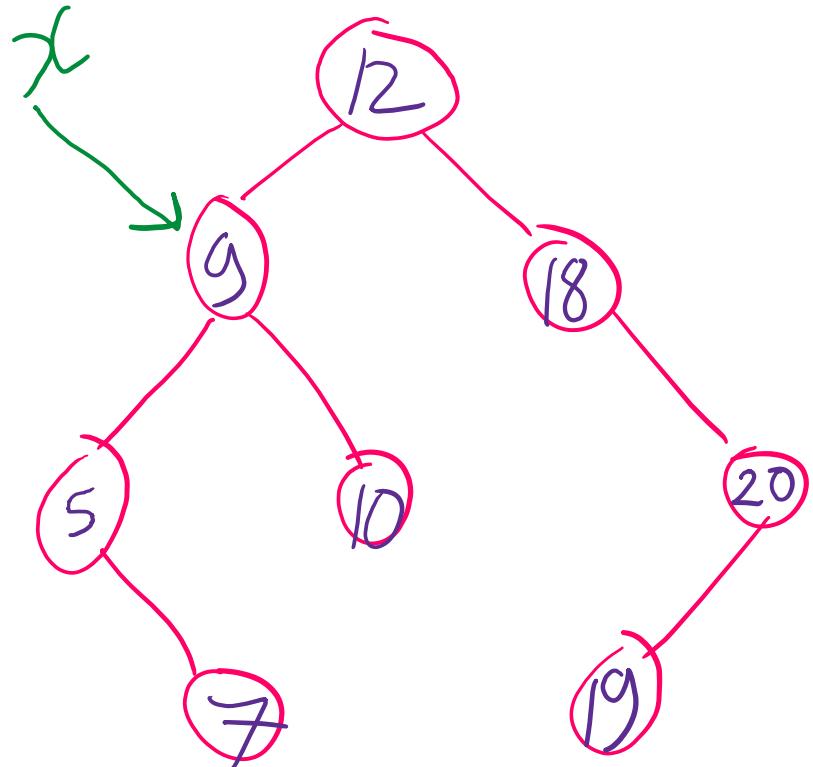
Visit order : 12

Iterative Preorder traversal



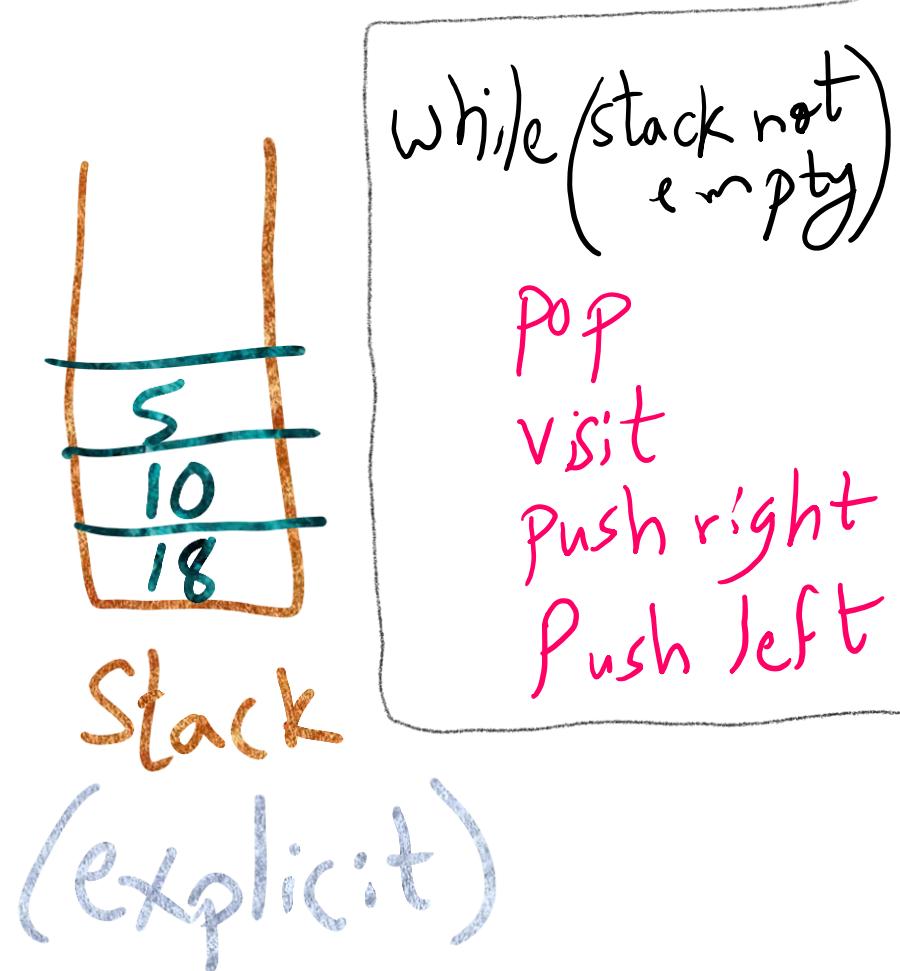
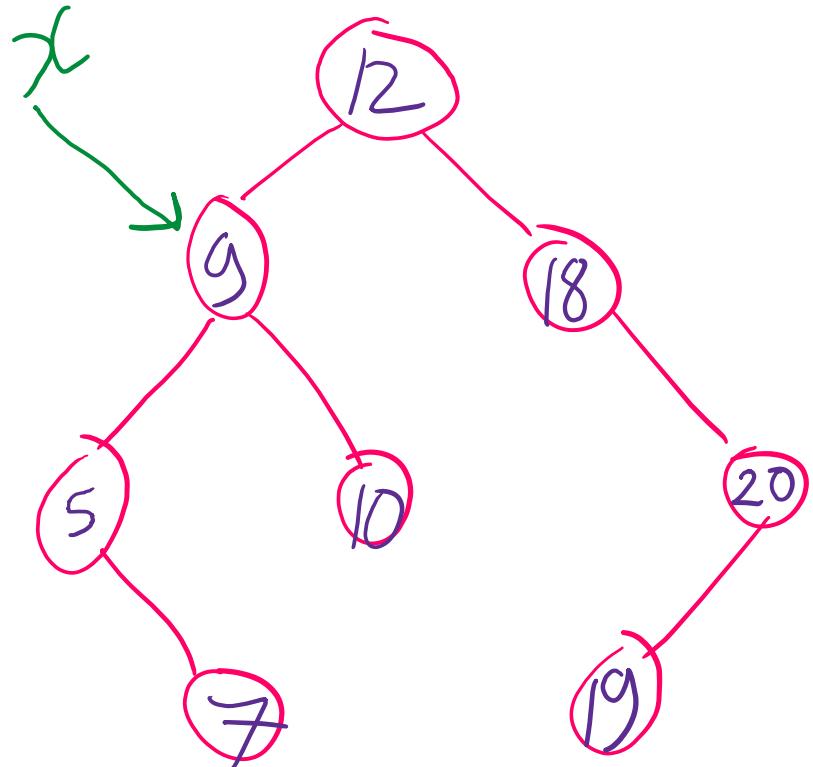
Visit order : 12

Iterative Preorder traversal



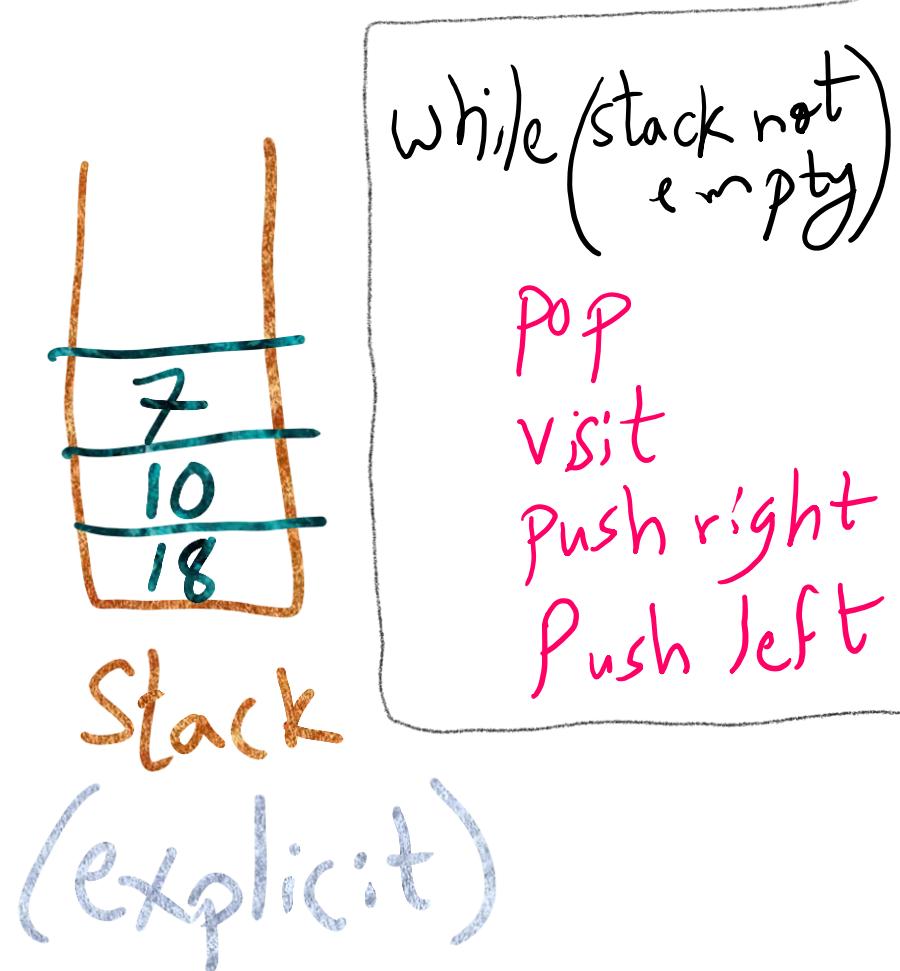
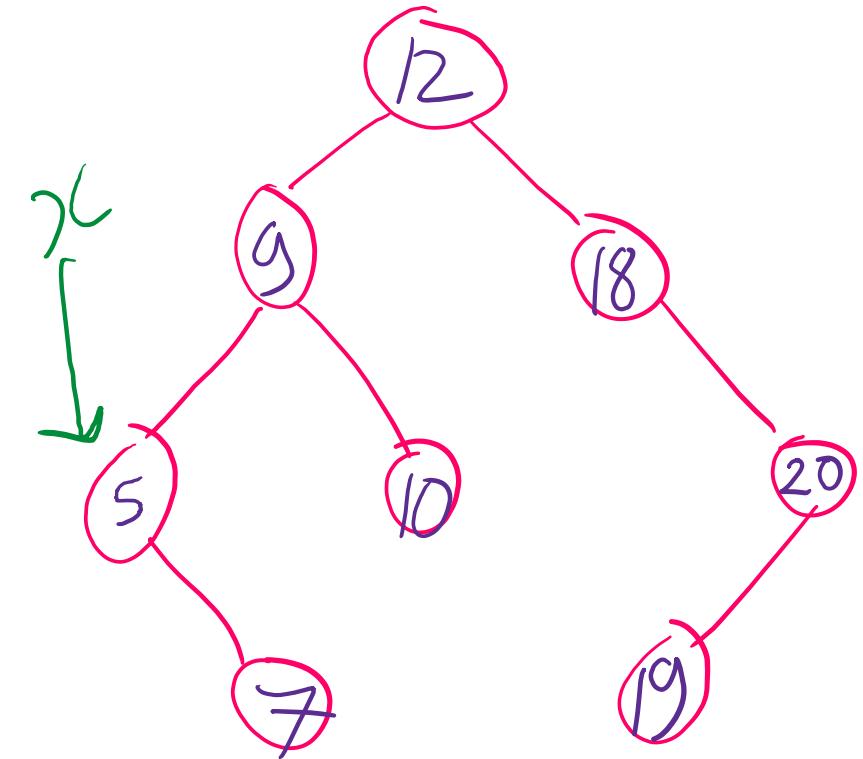
Visit order : 12, 9

Iterative Preorder traversal



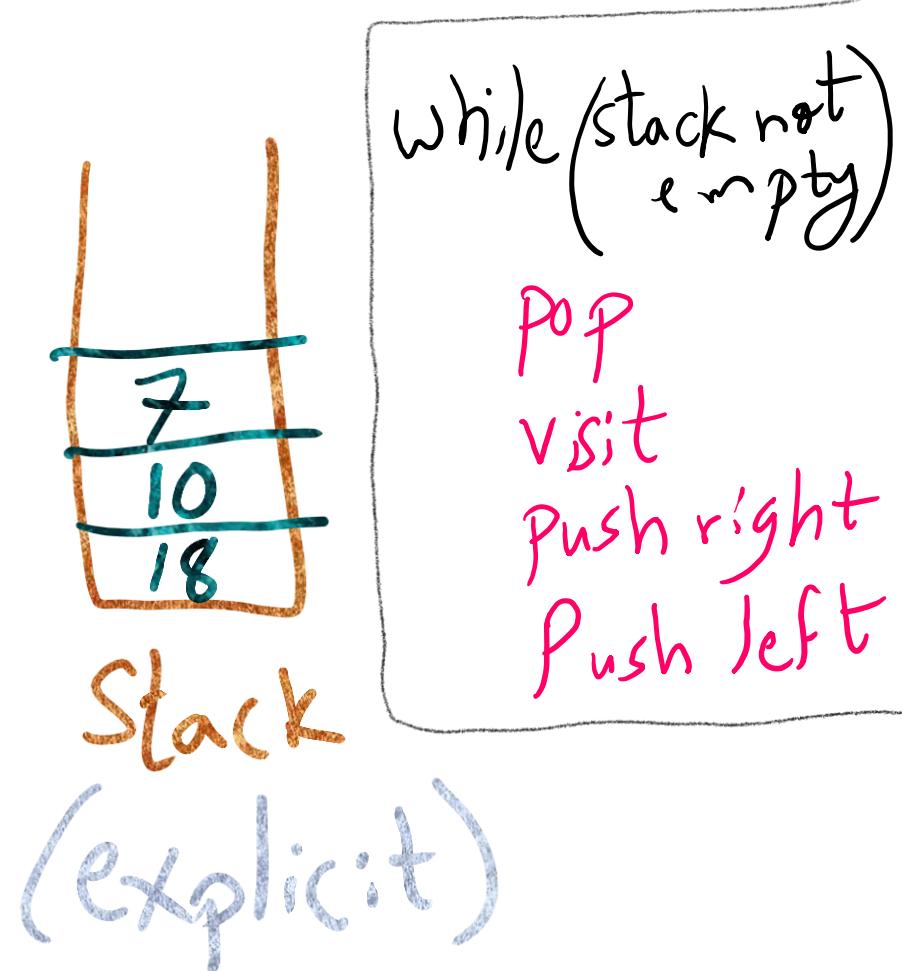
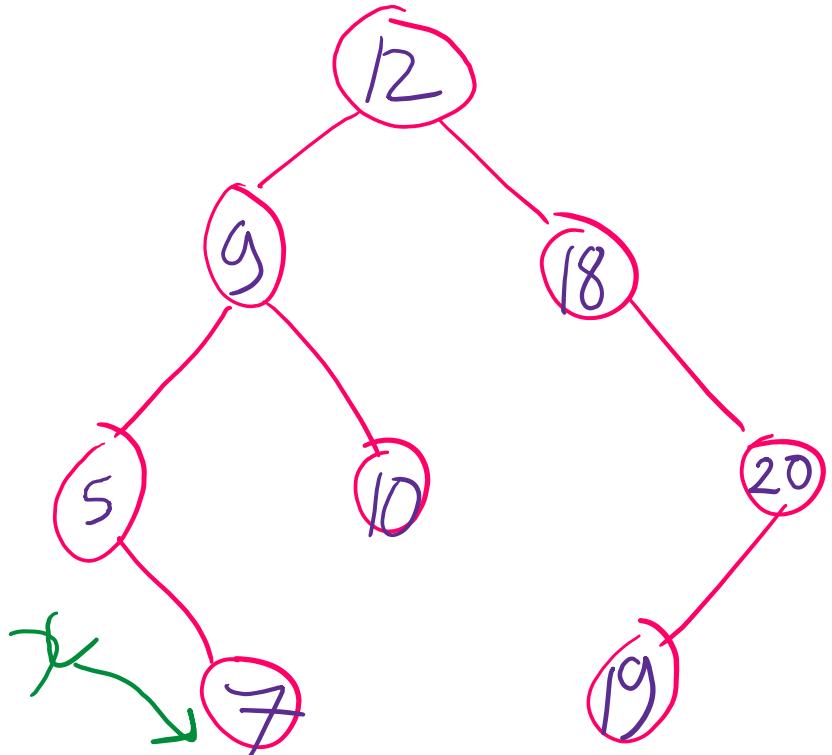
Visit order : 12, 9

Iterative Preorder traversal



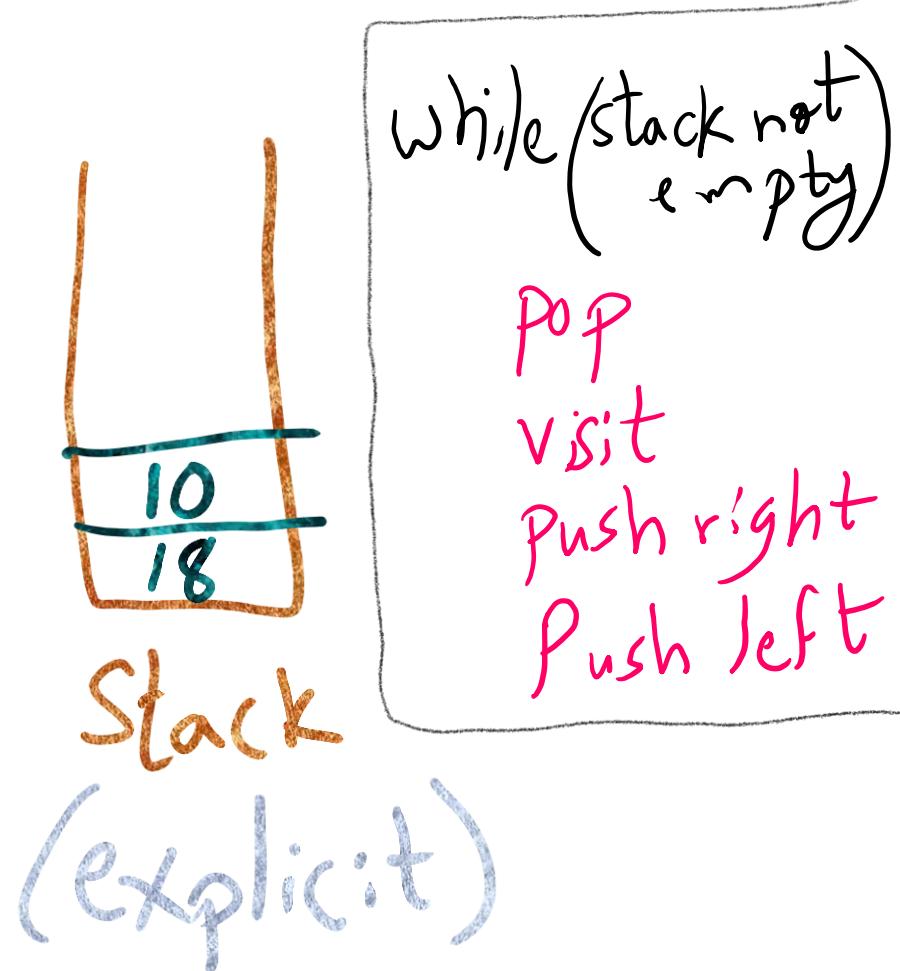
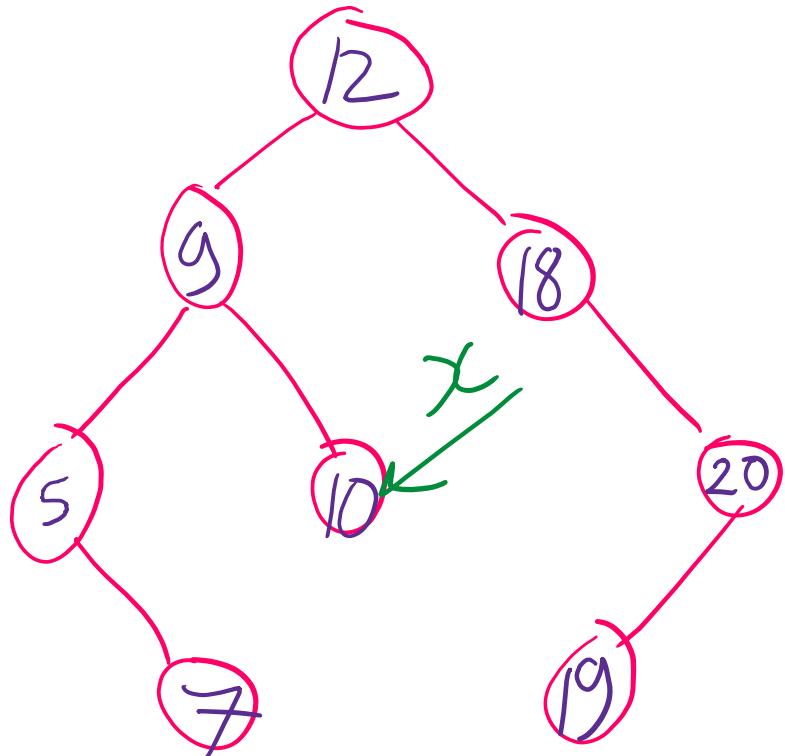
Visit order : 12, 9, 5

Iterative Preorder traversal



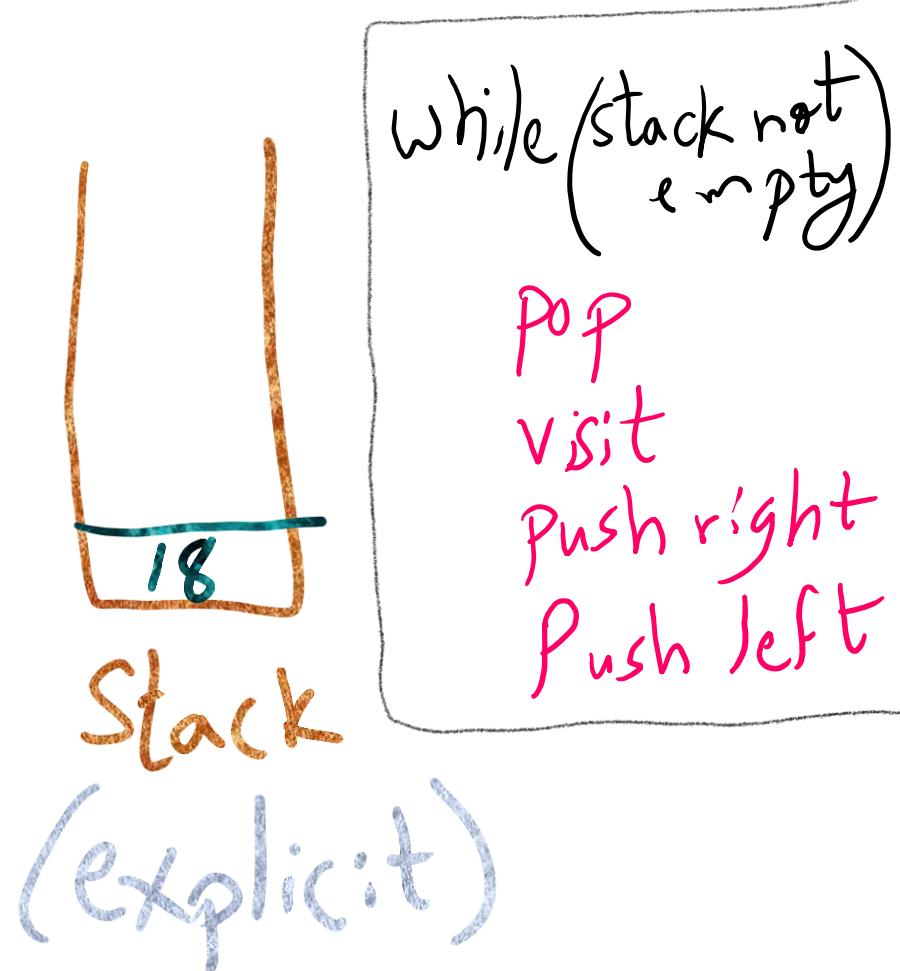
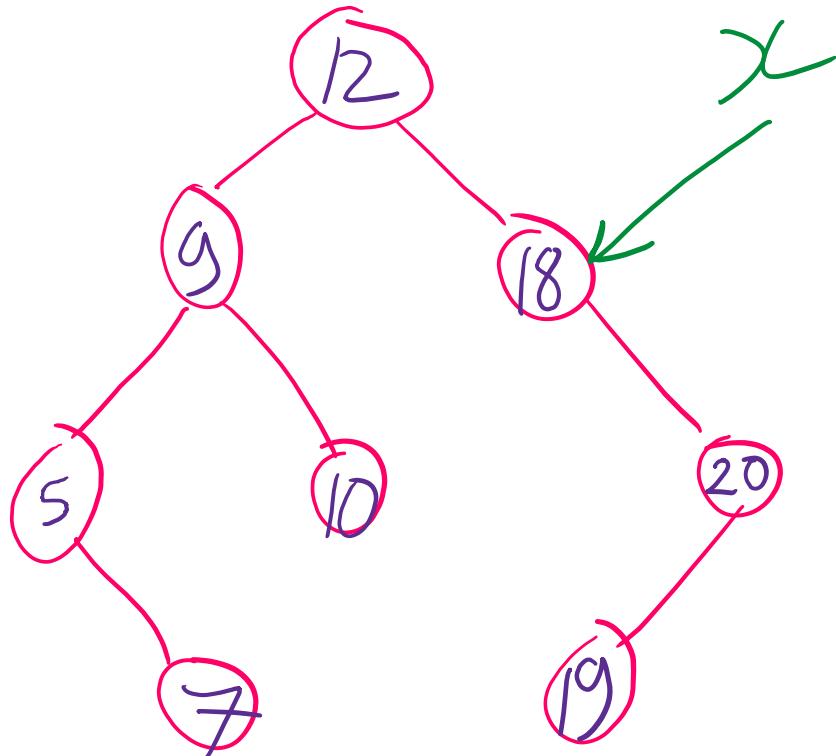
Visit order : 12, 9, 5, 7

Iterative Preorder traversal



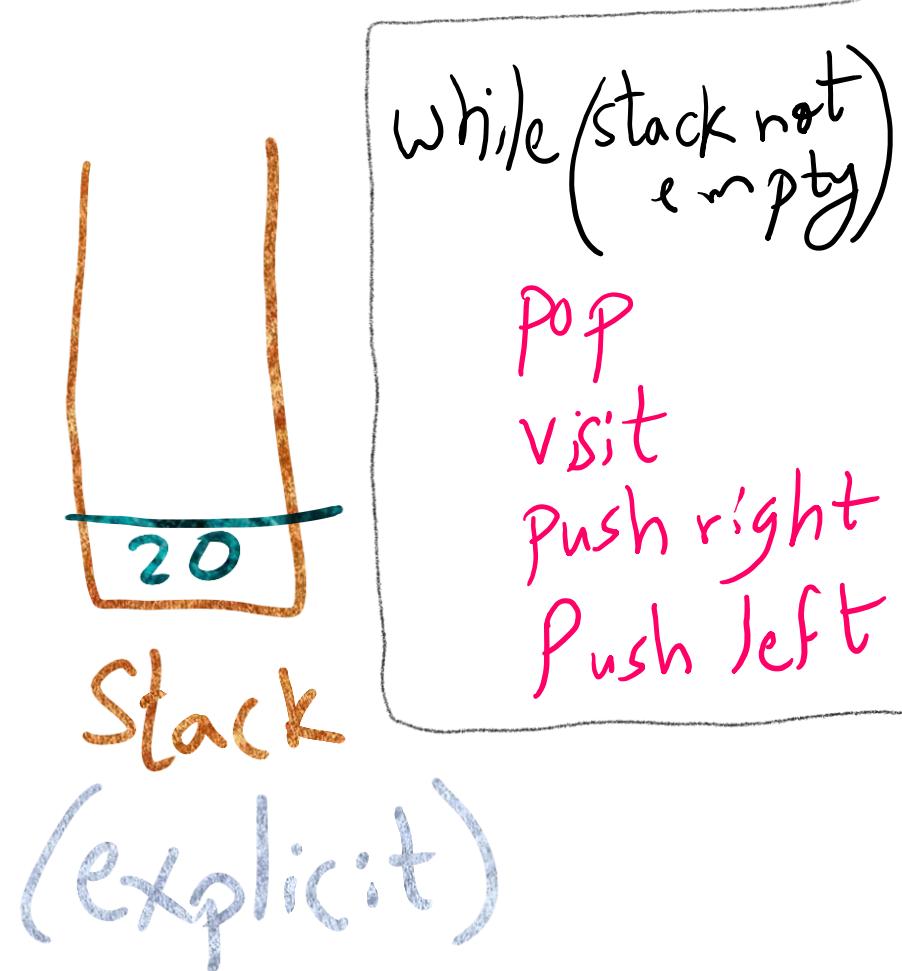
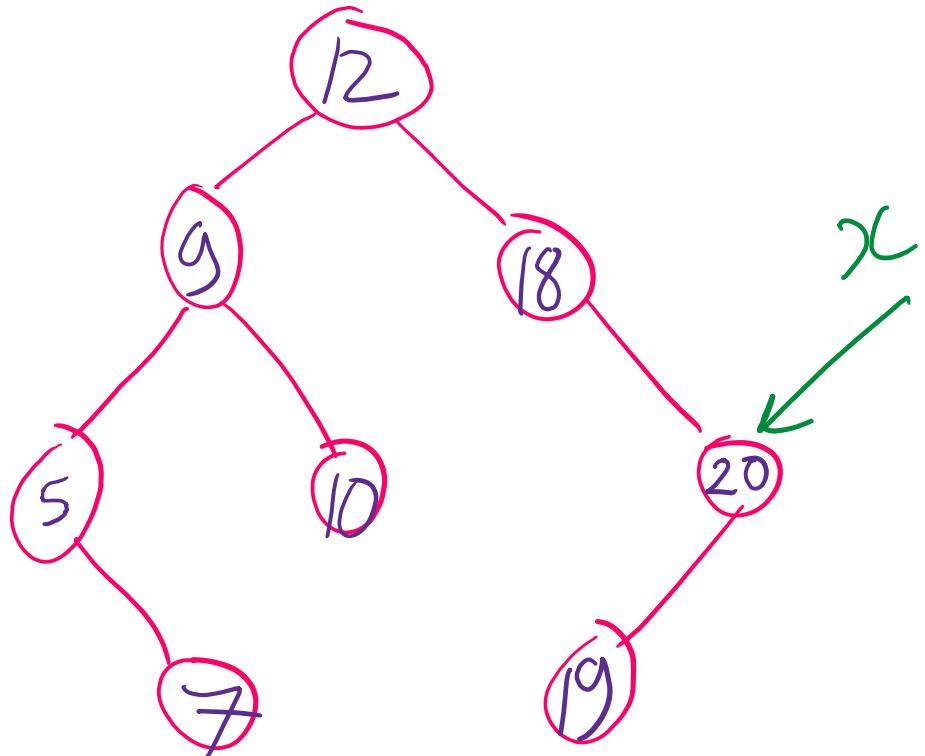
Visit order : 12, 9, 5, 7, 10

Iterative Preorder traversal



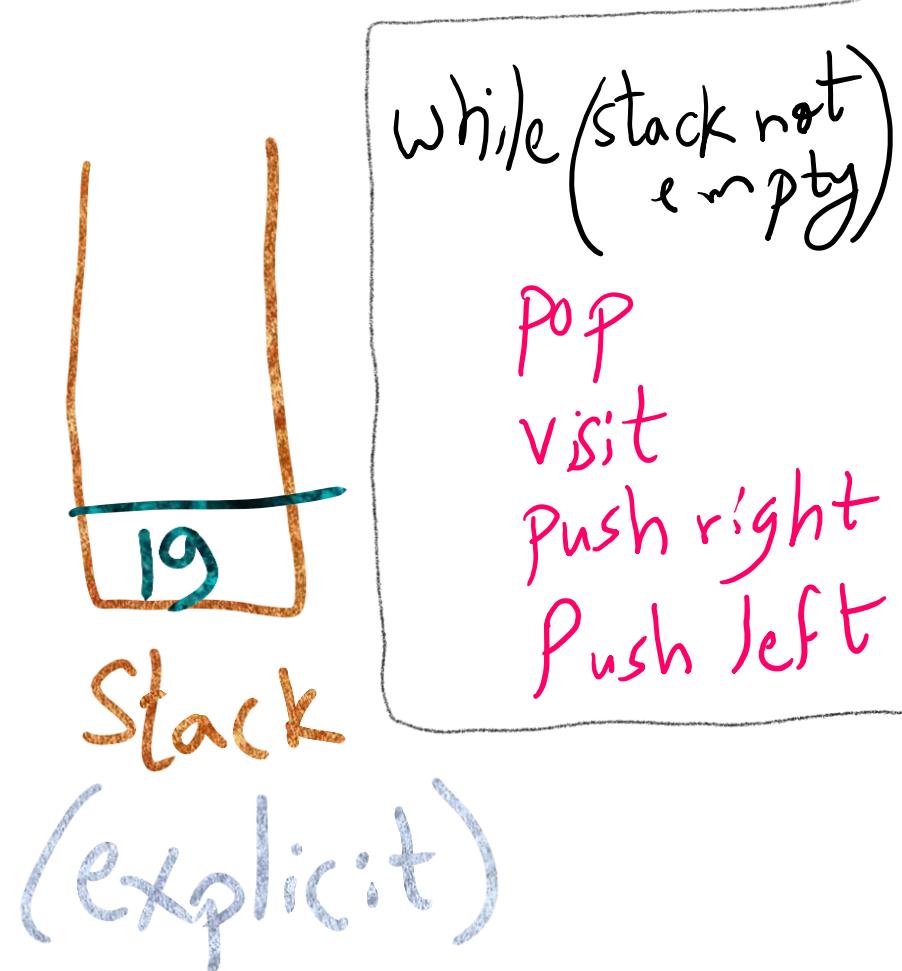
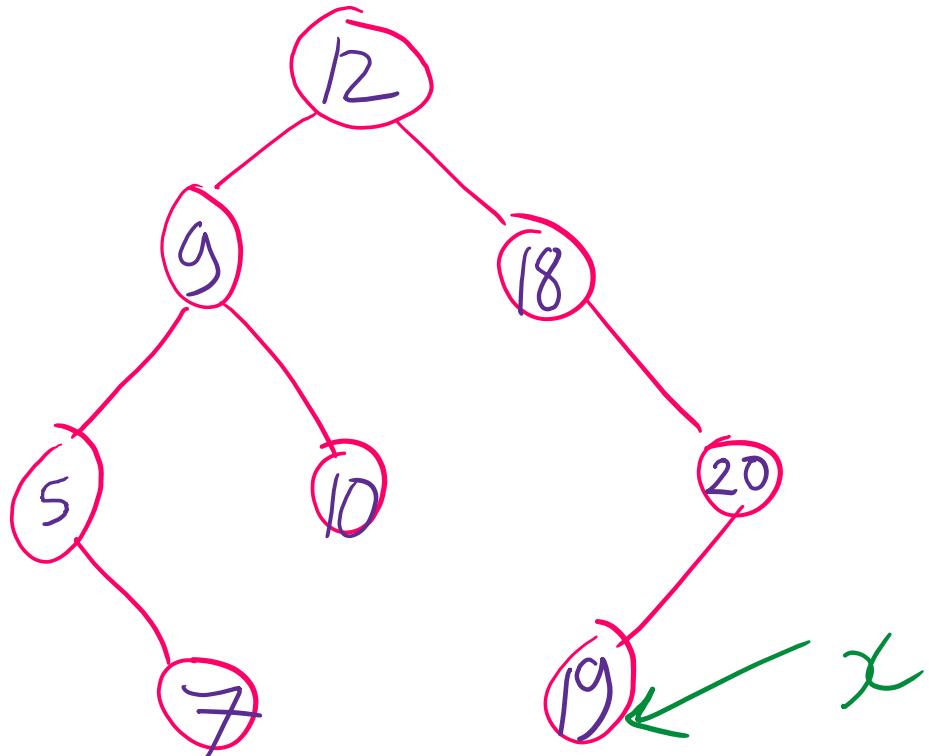
Visit order : 12, 9, 5, 7, 10, 18

Iterative Preorder traversal



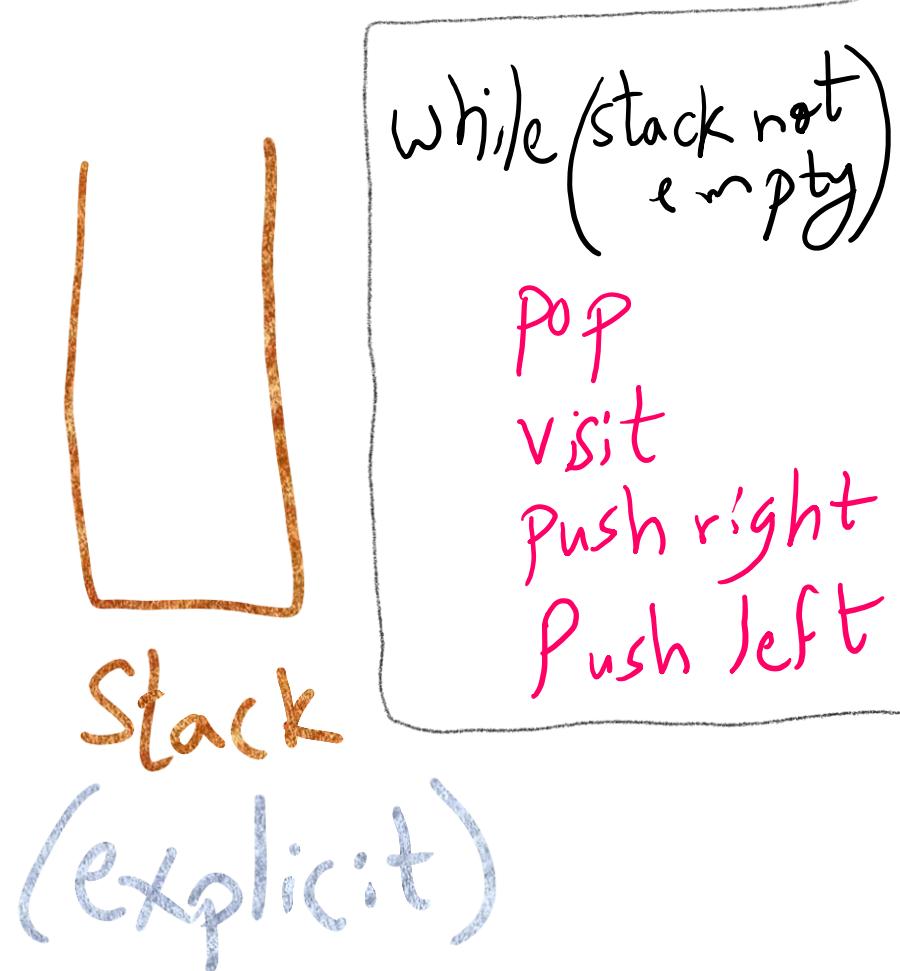
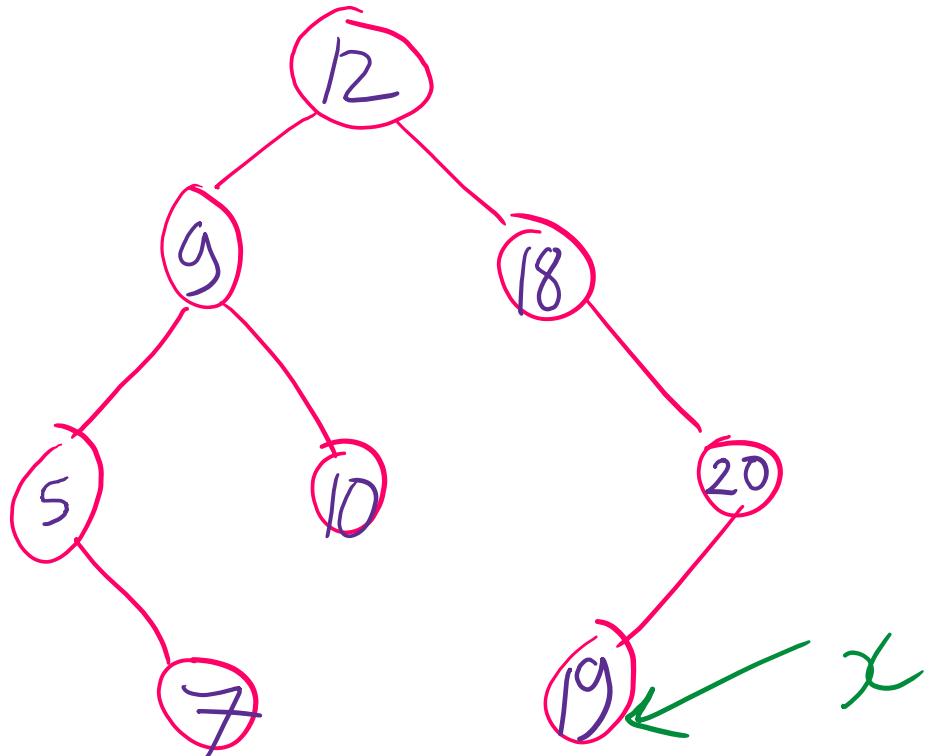
Visit order : 12, 9, 5, 7, 10, 18, 20

Iterative Preorder traversal



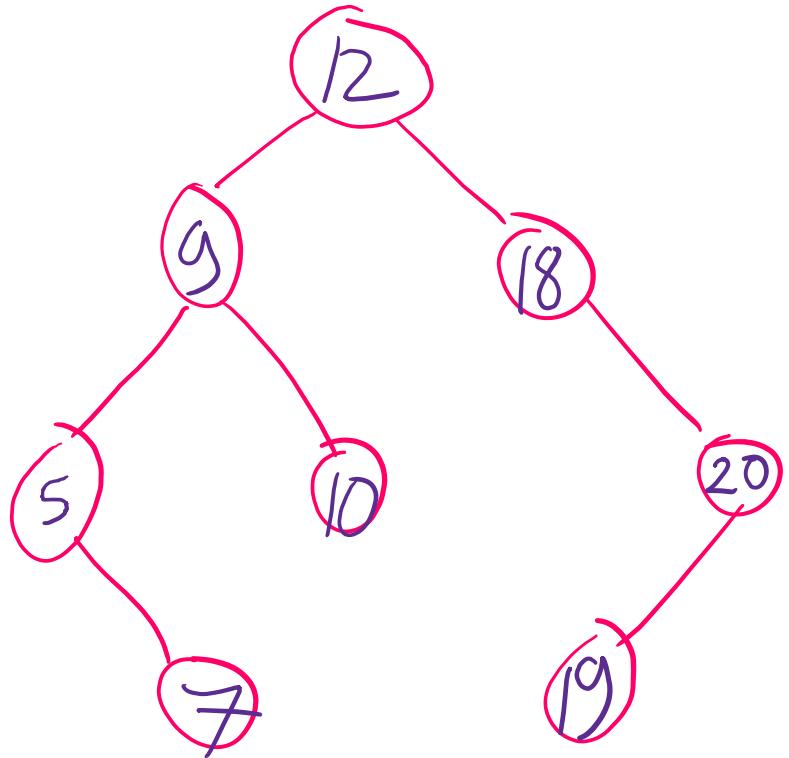
Visit order : 12, 9, 5, 7, 10, 18, 20, 19

Iterative Preorder traversal



Visit order : 12, 9, 5, 7, 10, 18, 20, 19

Iterative Inorder traversal



Visit order :

repeat until stack empty & $x == \text{null}$

$x = \text{leftmost node}$

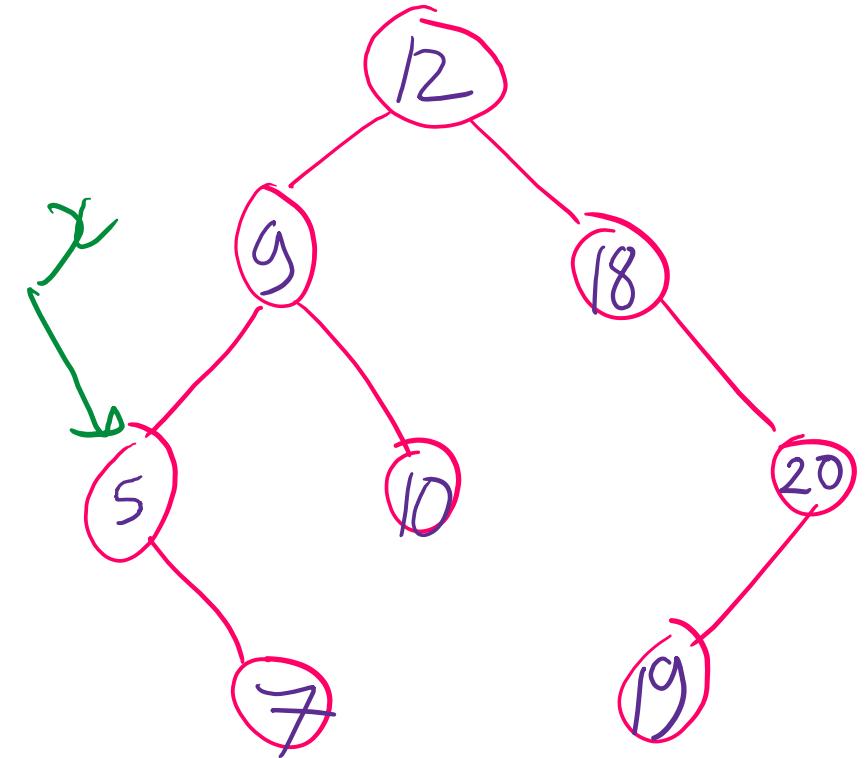
push to stack while moving down

pop & visit

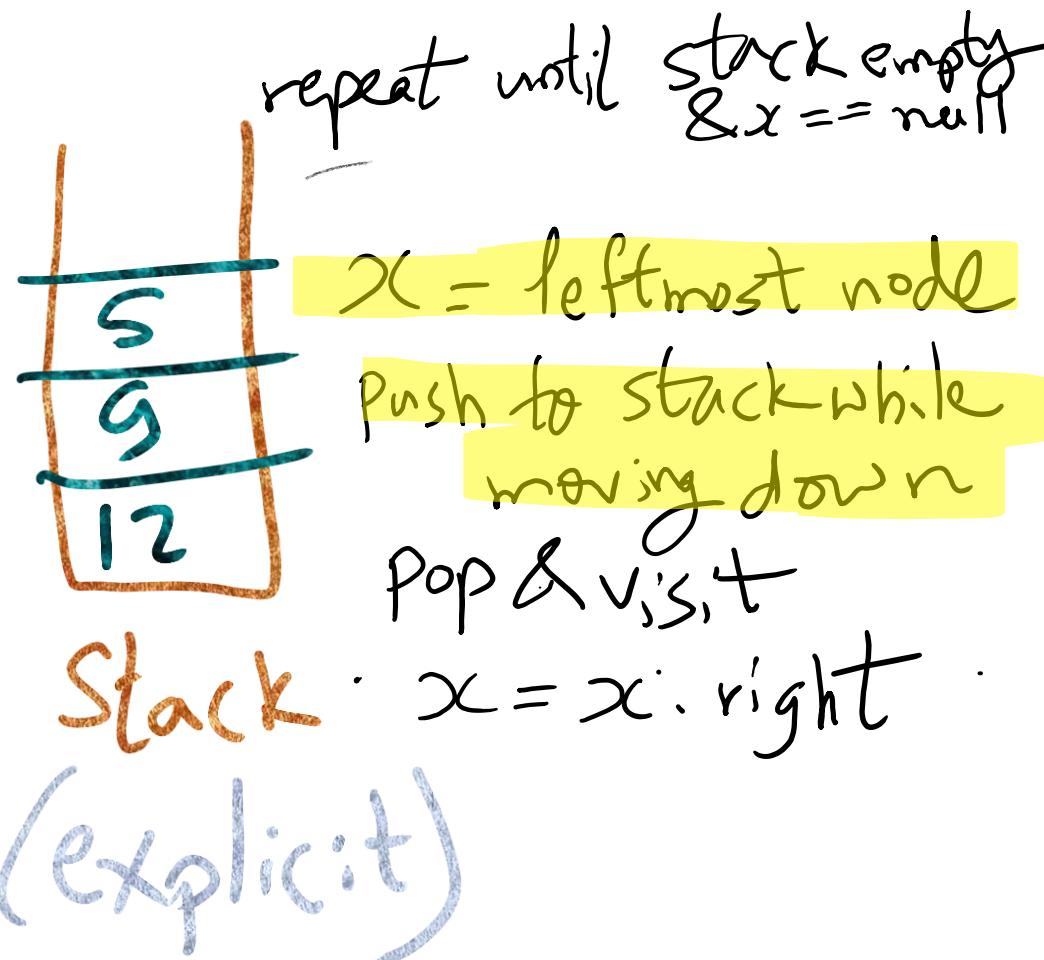
$x = x.\text{right}$

Stack (explicit)

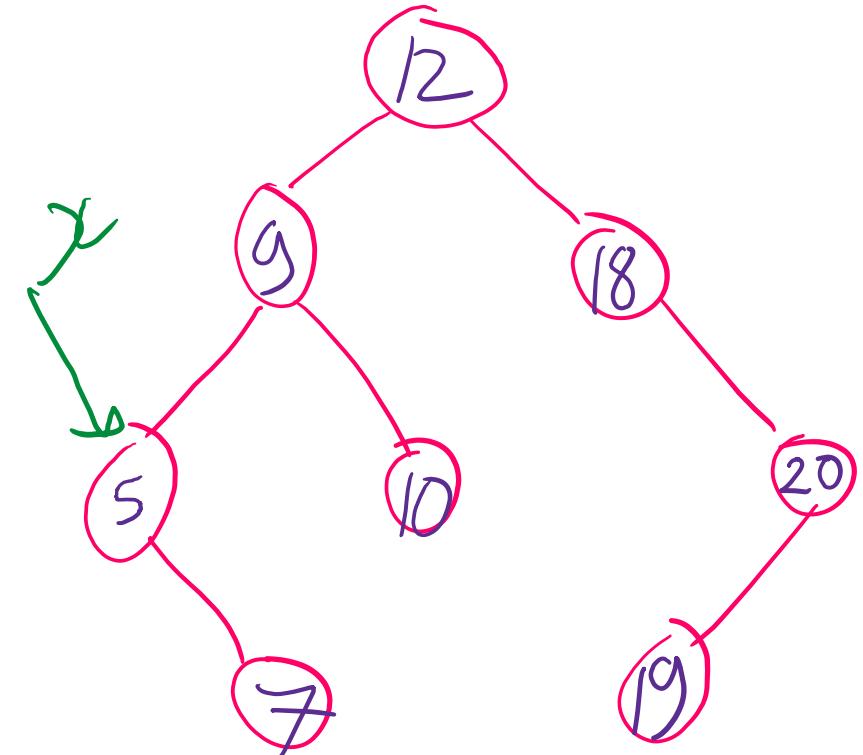
Iterative Inorder traversal



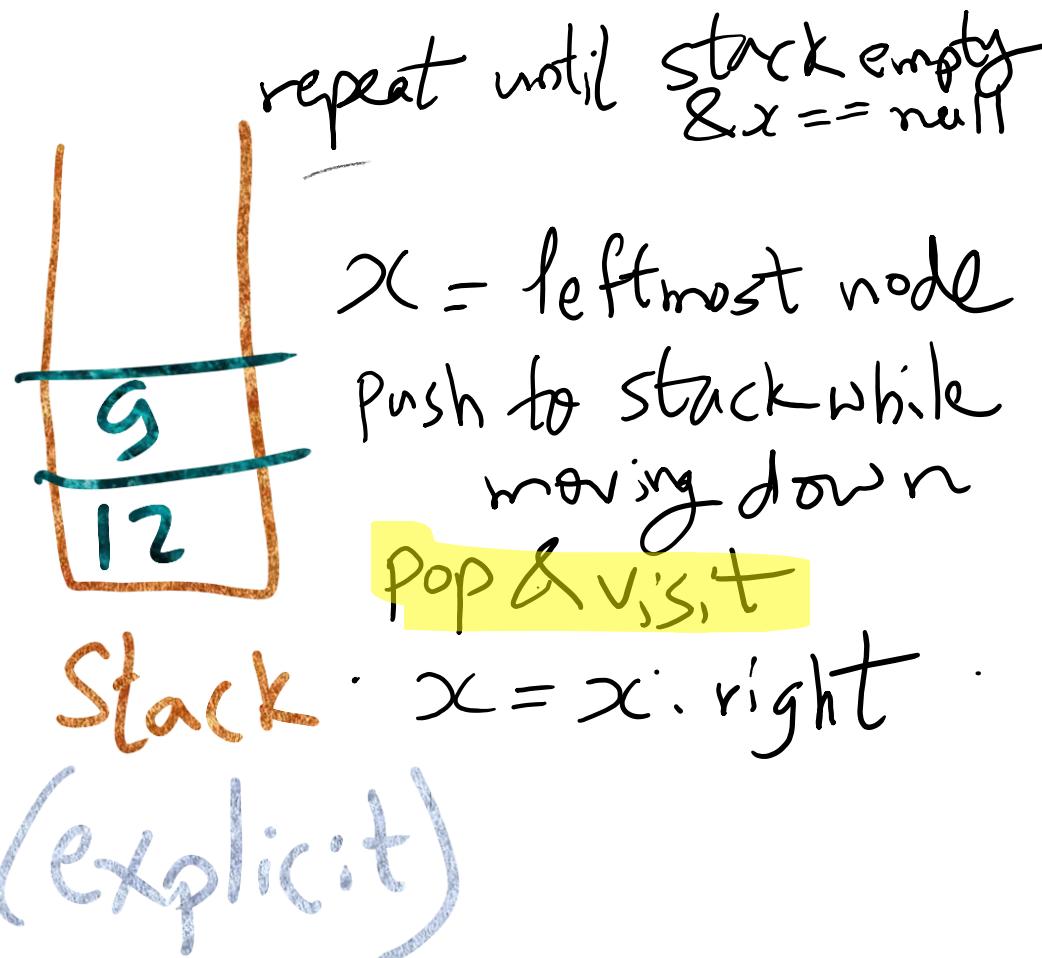
Visit order :



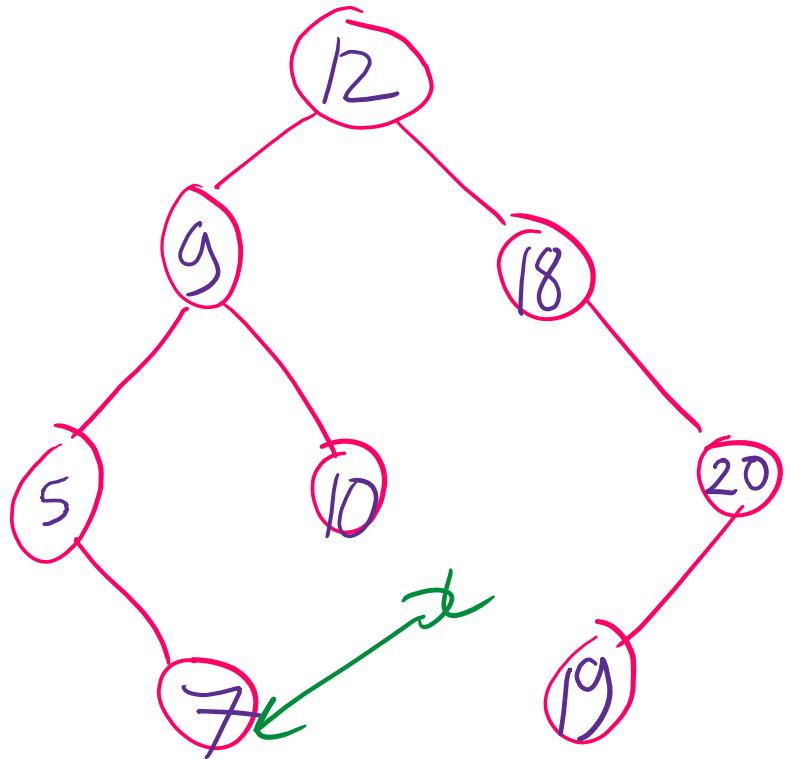
Iterative Inorder traversal



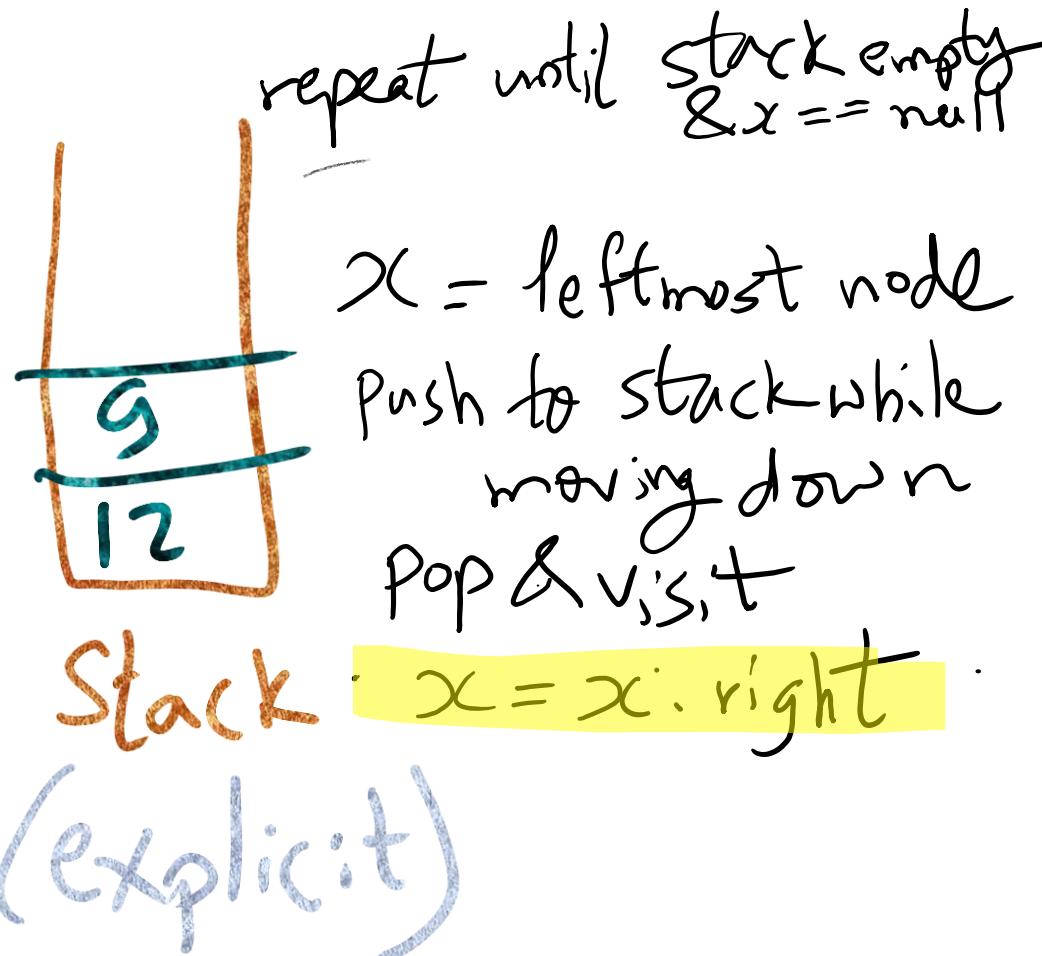
Visit order : 5



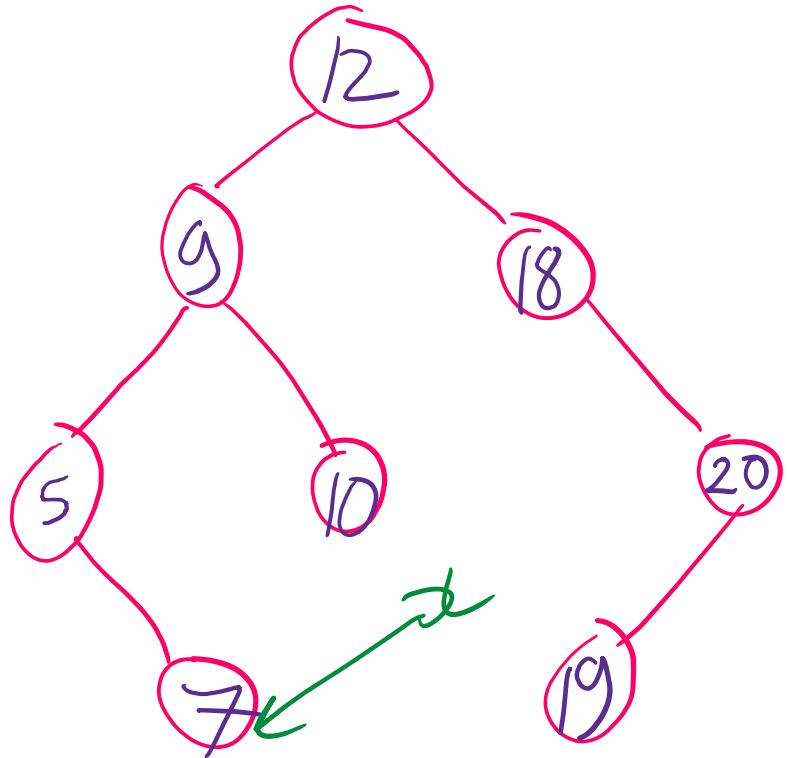
Iterative Inorder traversal



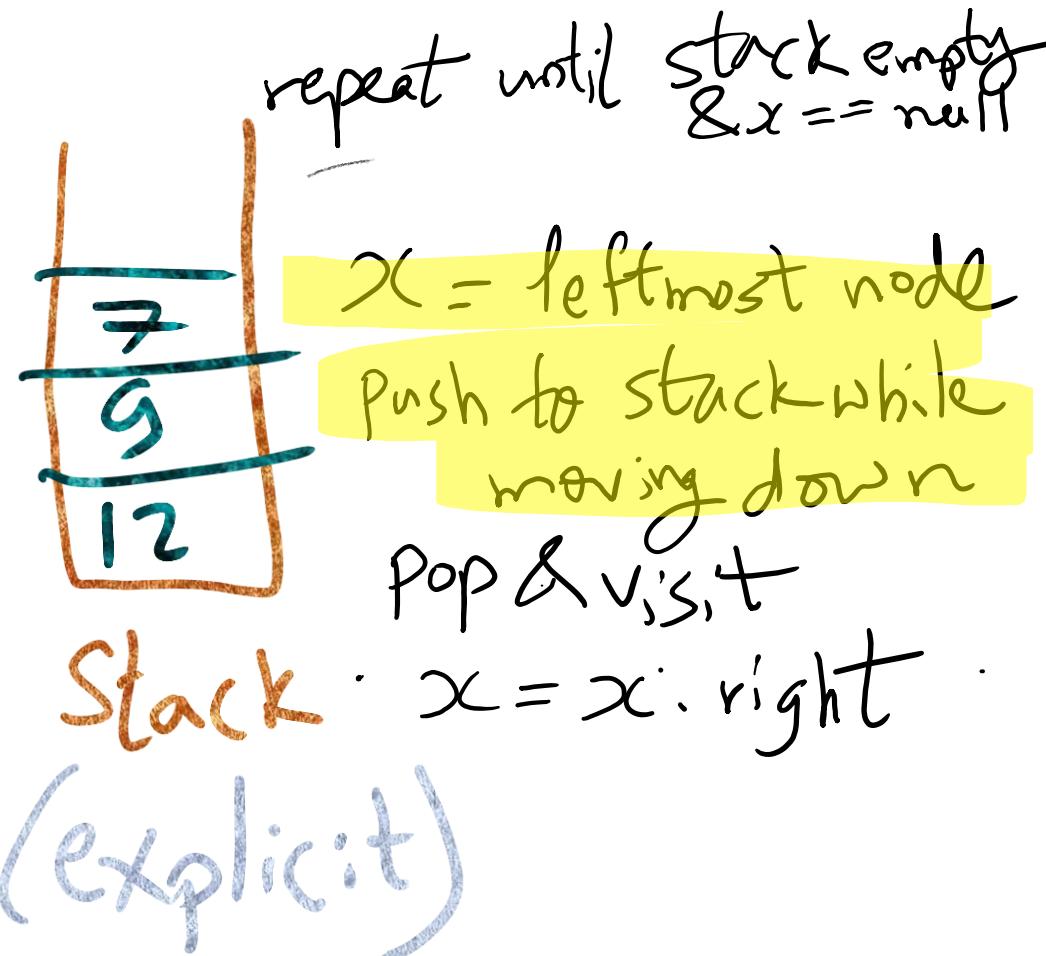
Visit order : 5



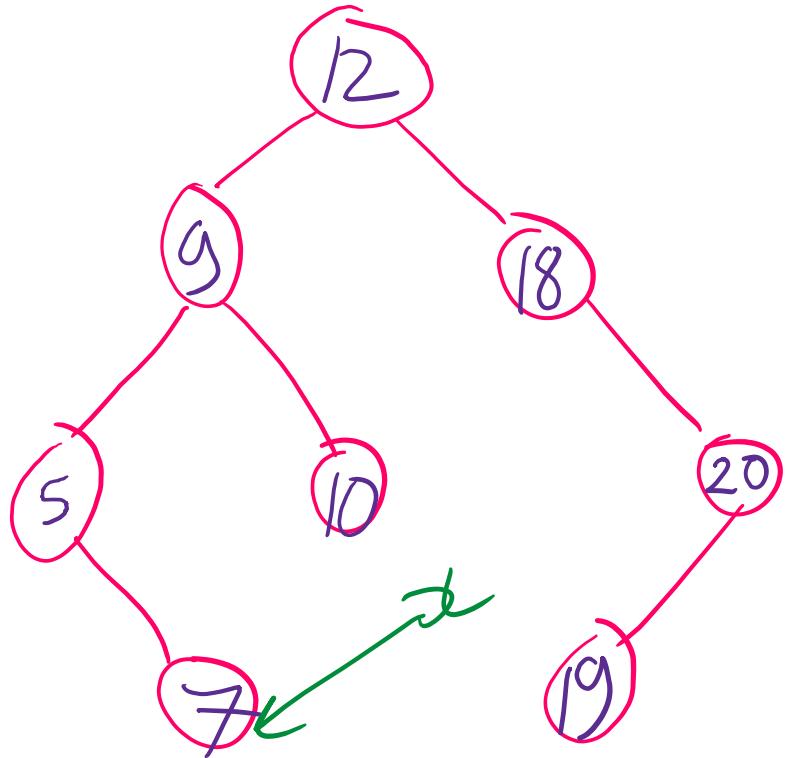
Iterative Inorder traversal



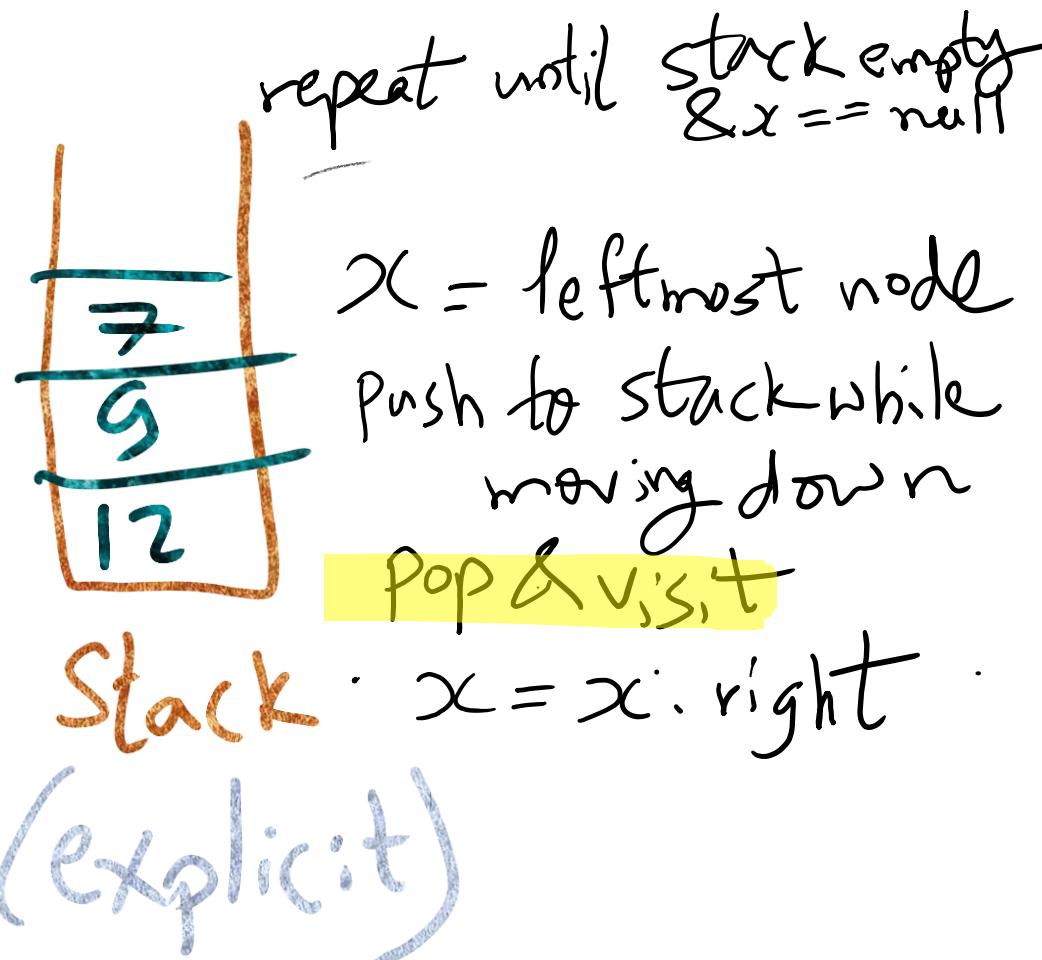
Visit order : 5



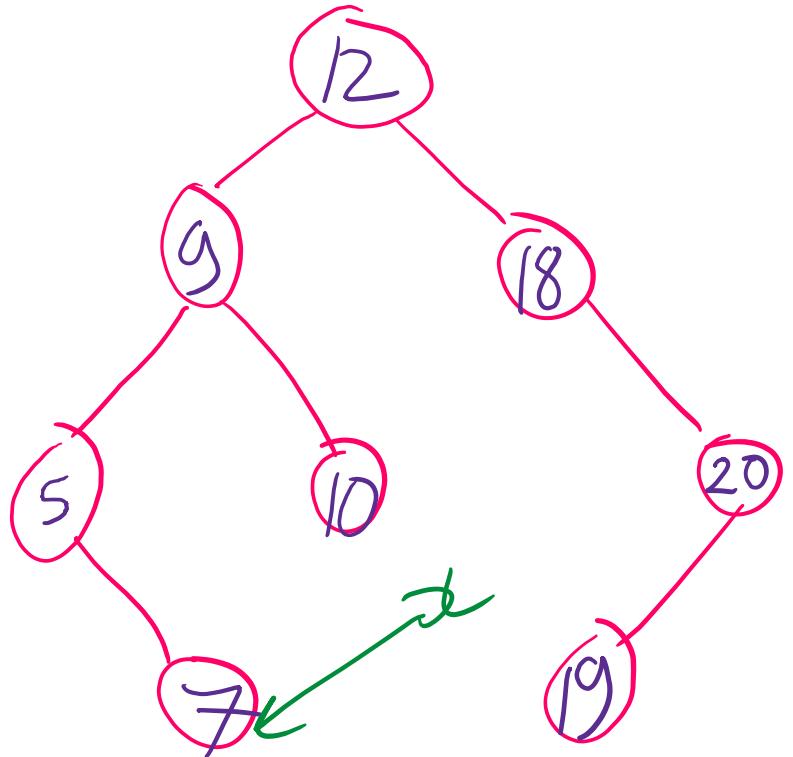
Iterative Inorder traversal



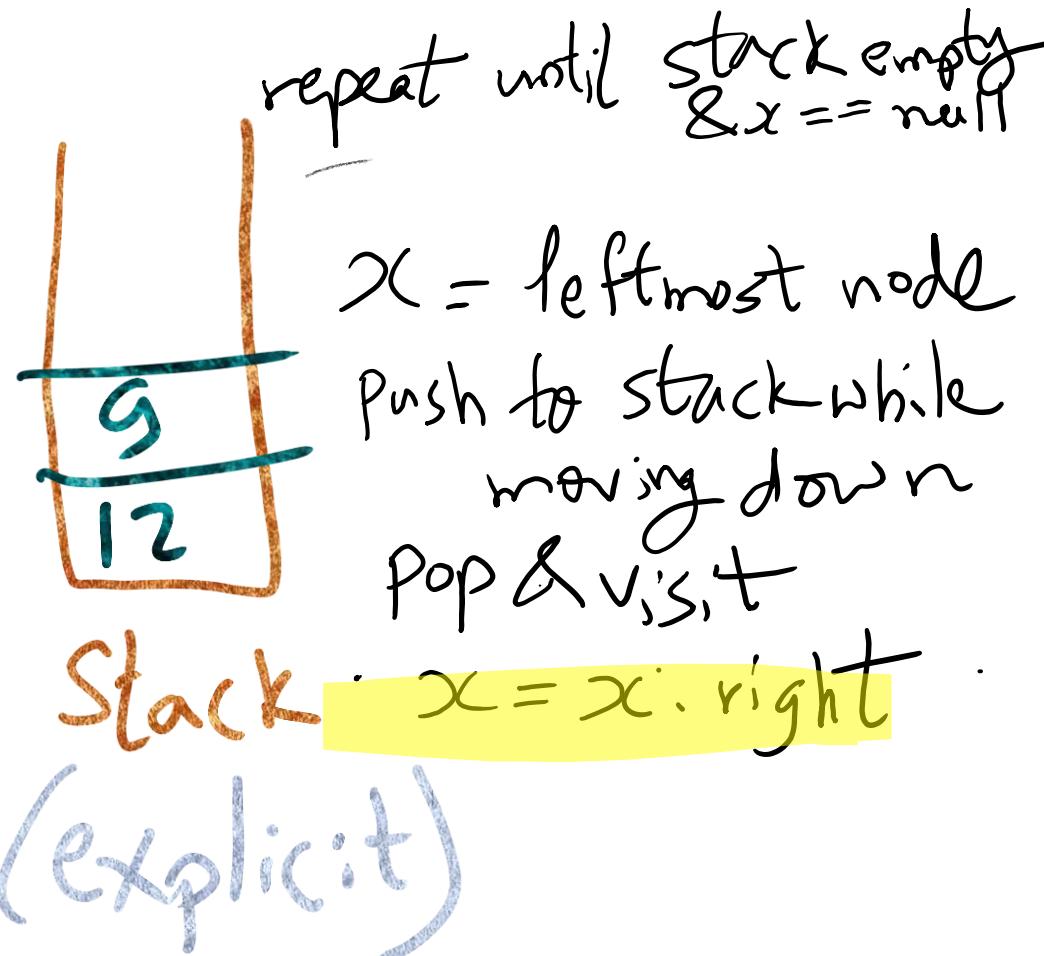
Visit order : 5, 7



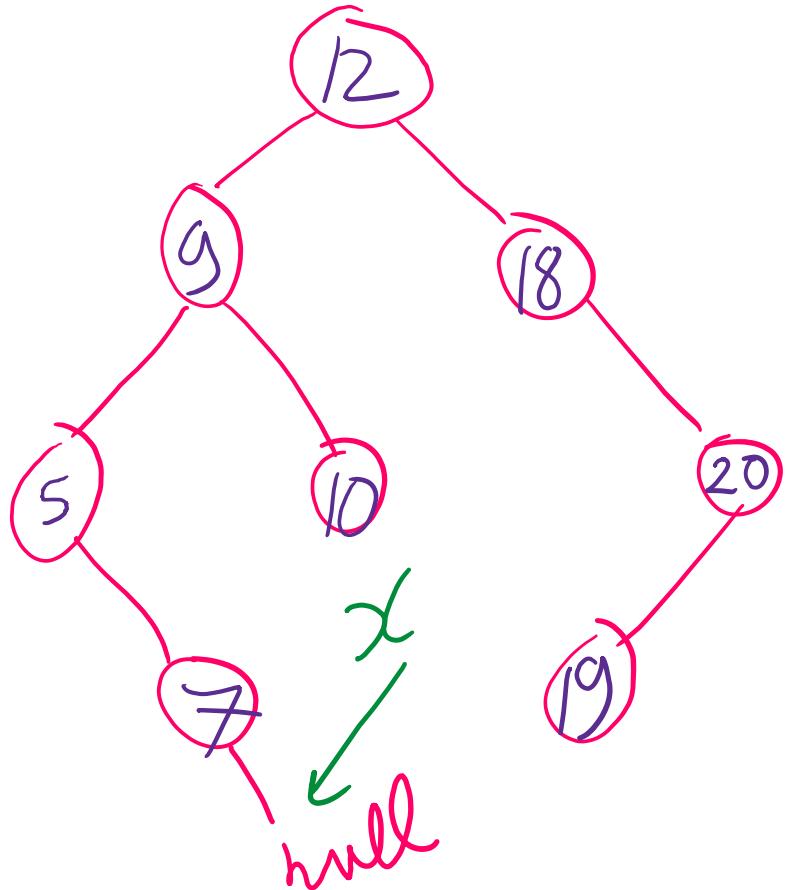
Iterative Inorder traversal



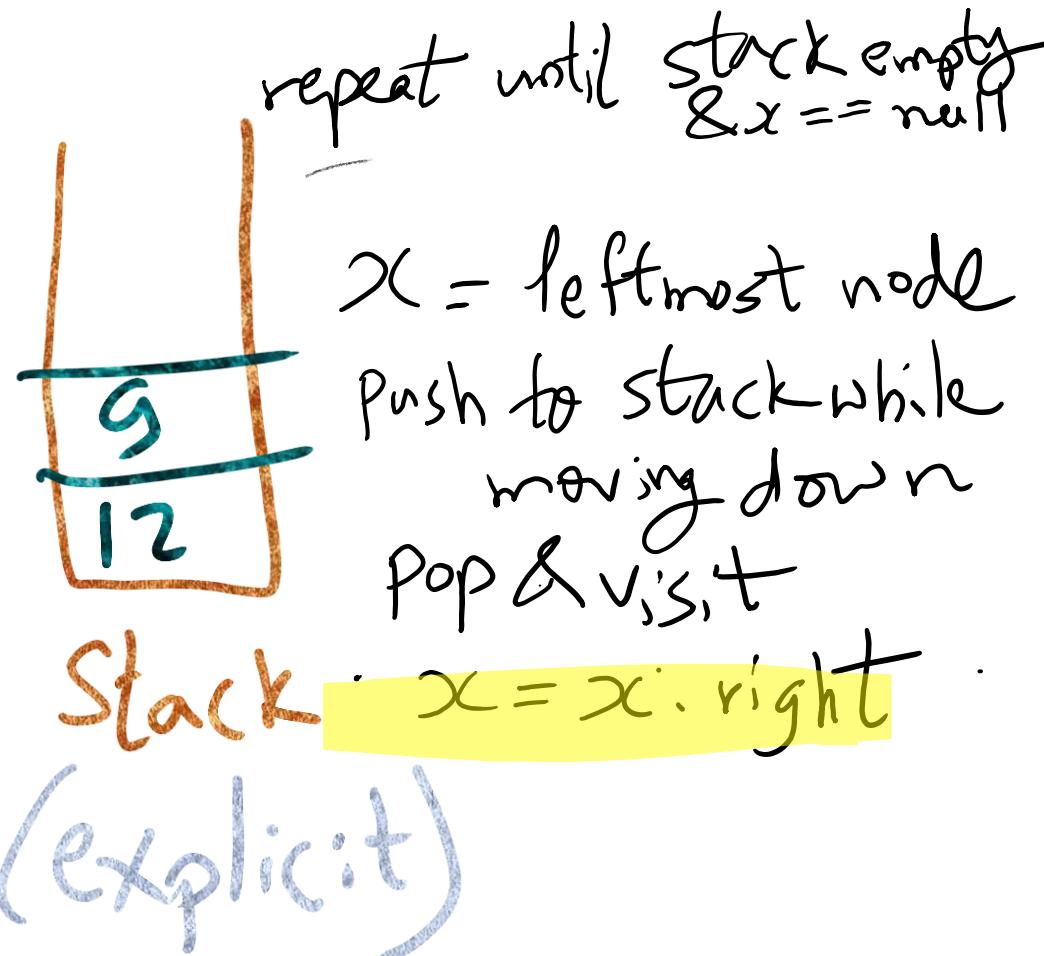
Visit order : 5, 7



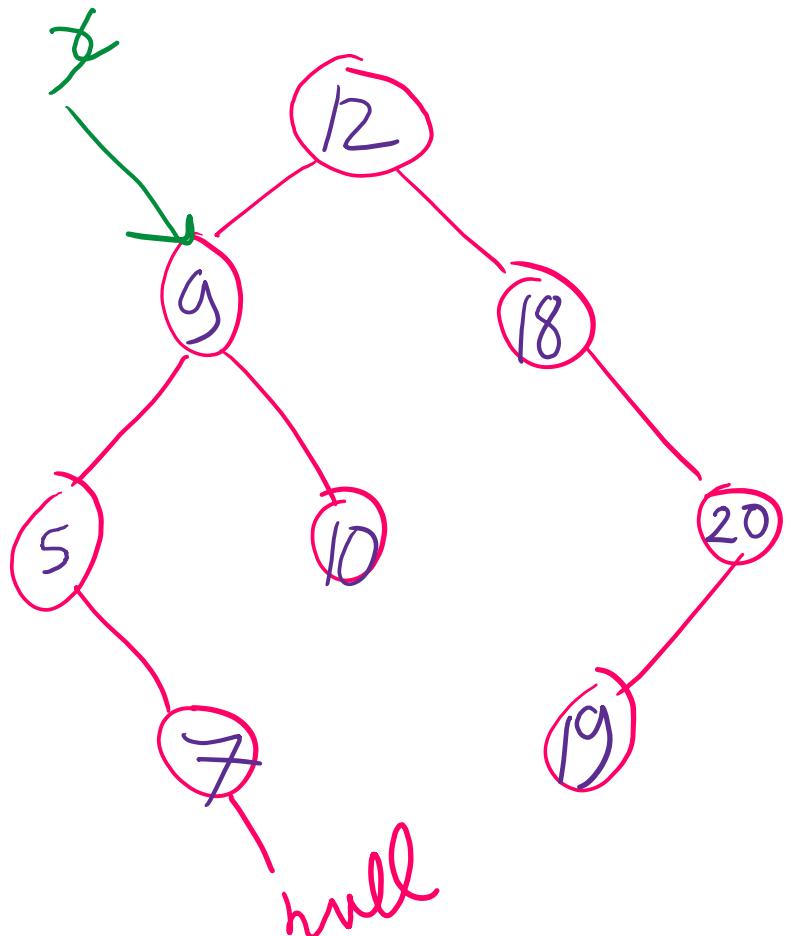
Iterative Inorder traversal



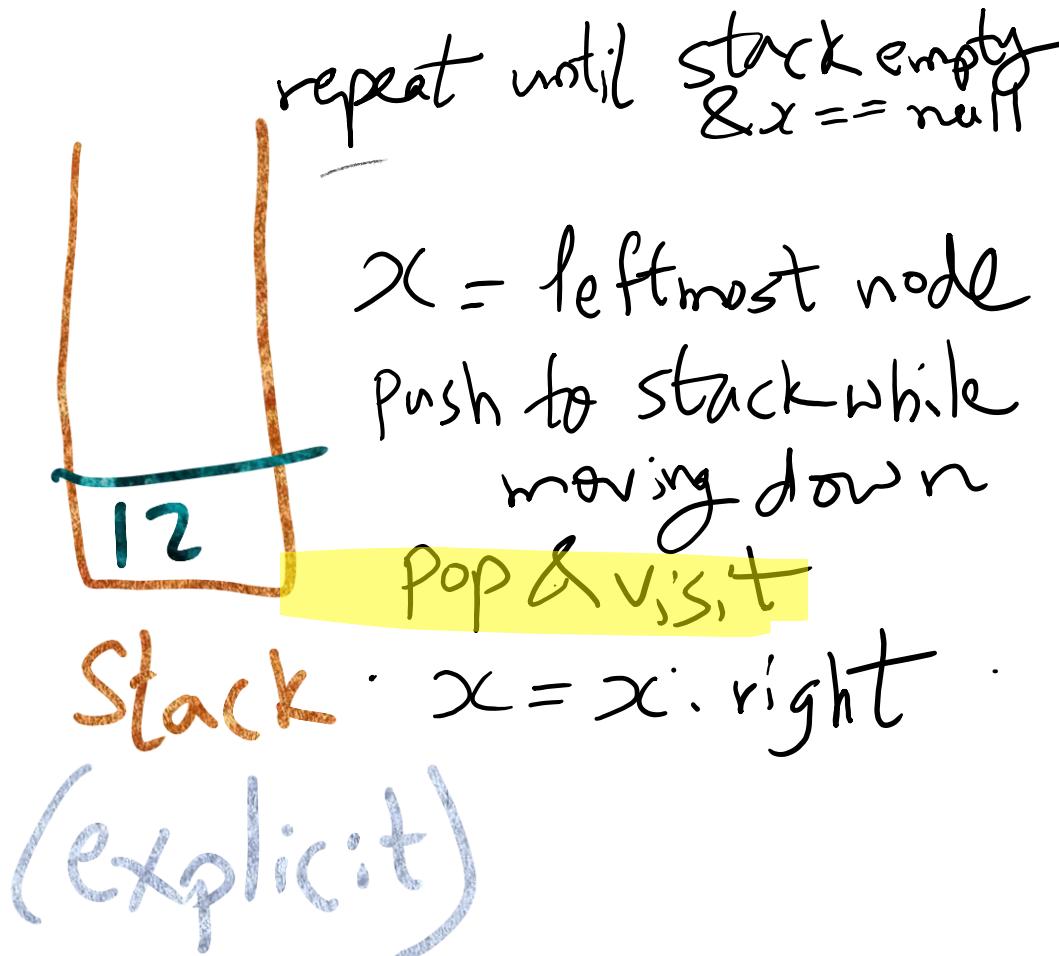
Visit order : 5, 7



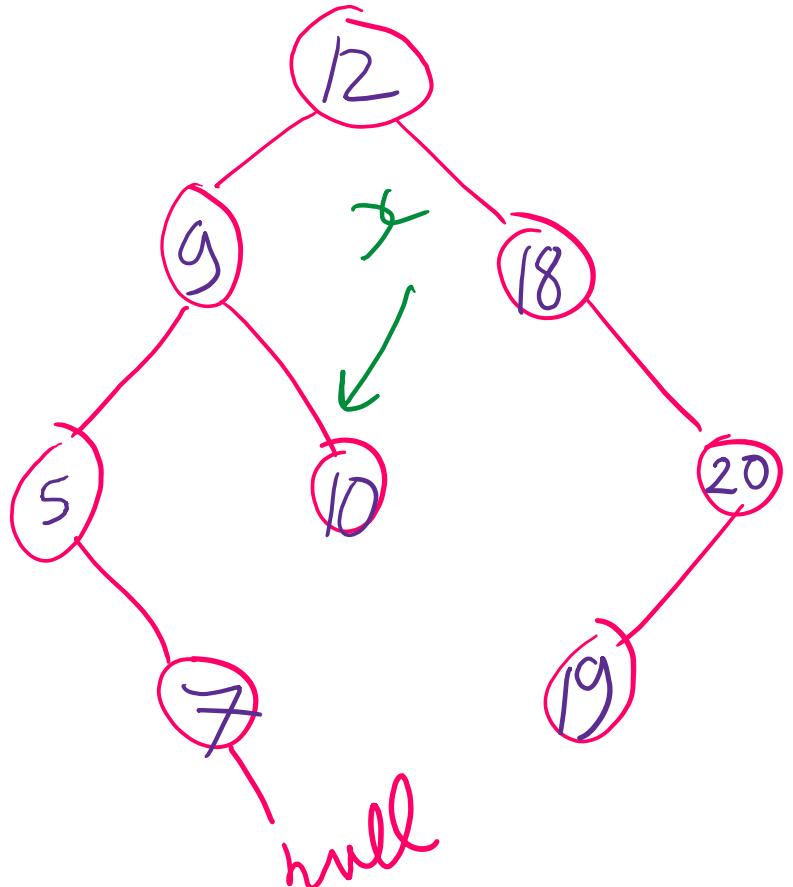
Iterative Inorder traversal



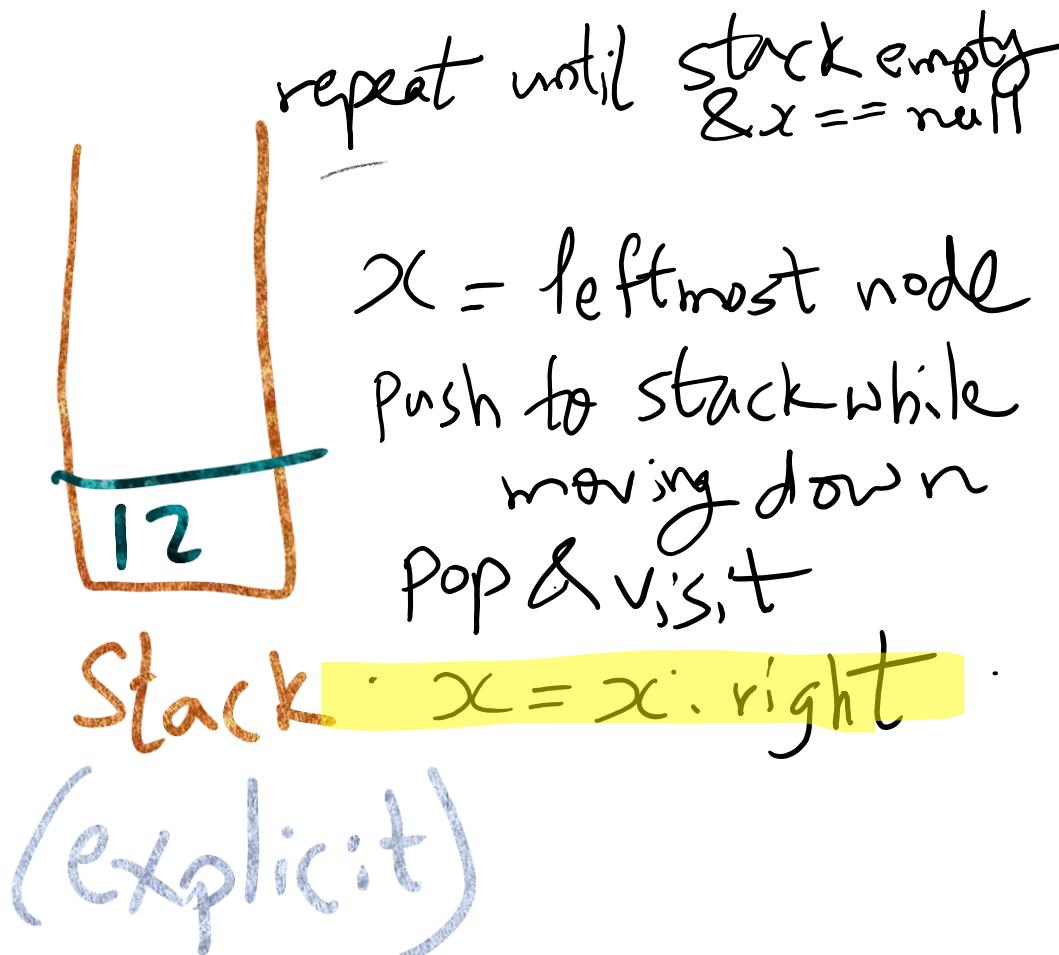
Visit order : 5, 7, 9



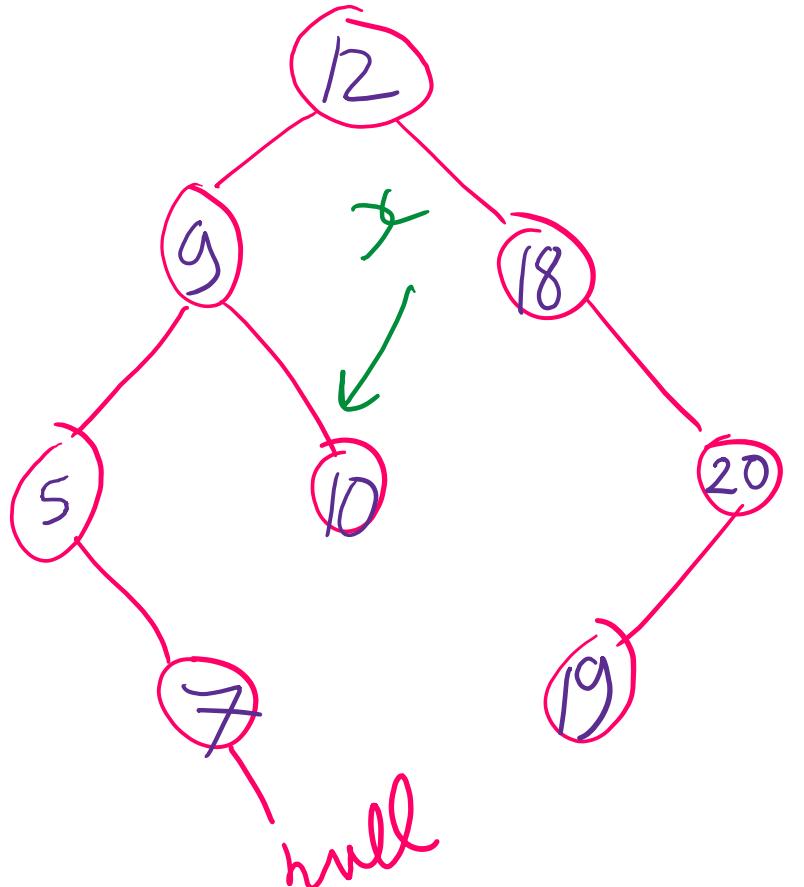
Iterative Inorder traversal



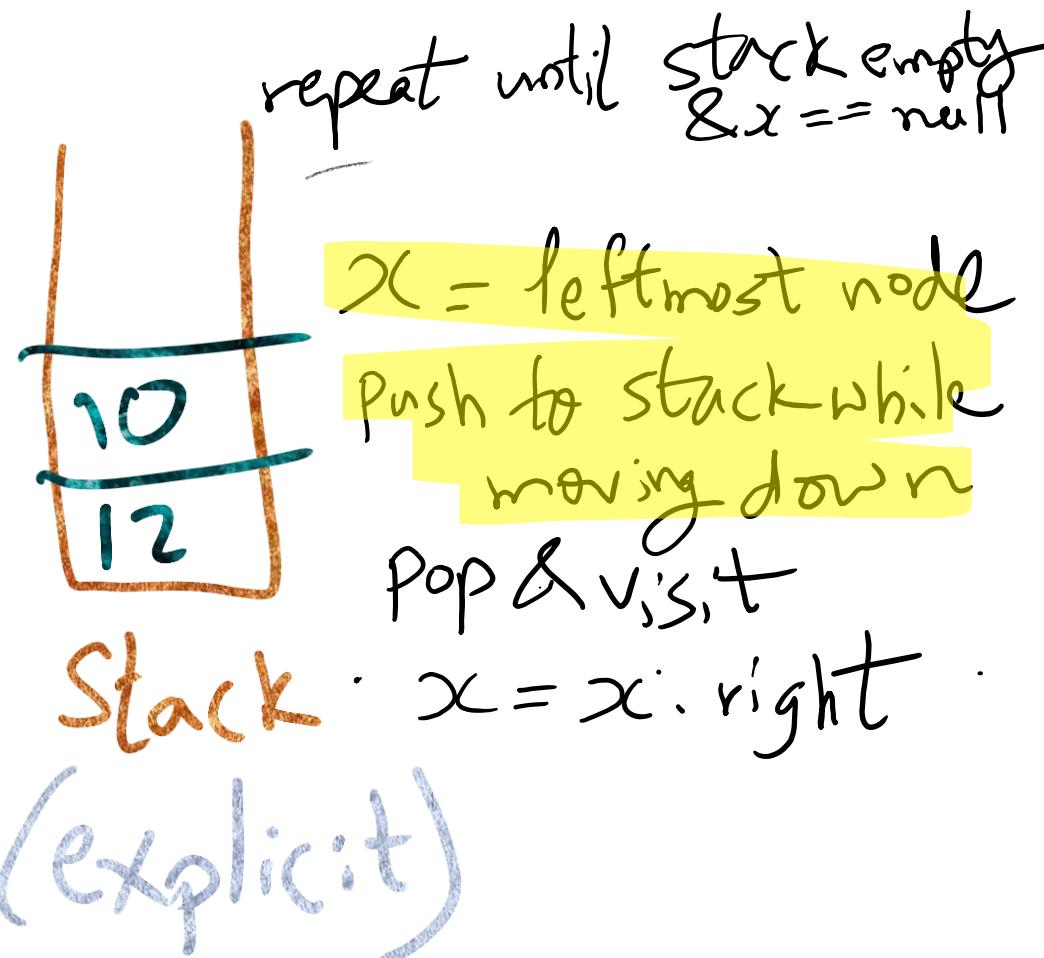
Visit order : 5, 7, 9



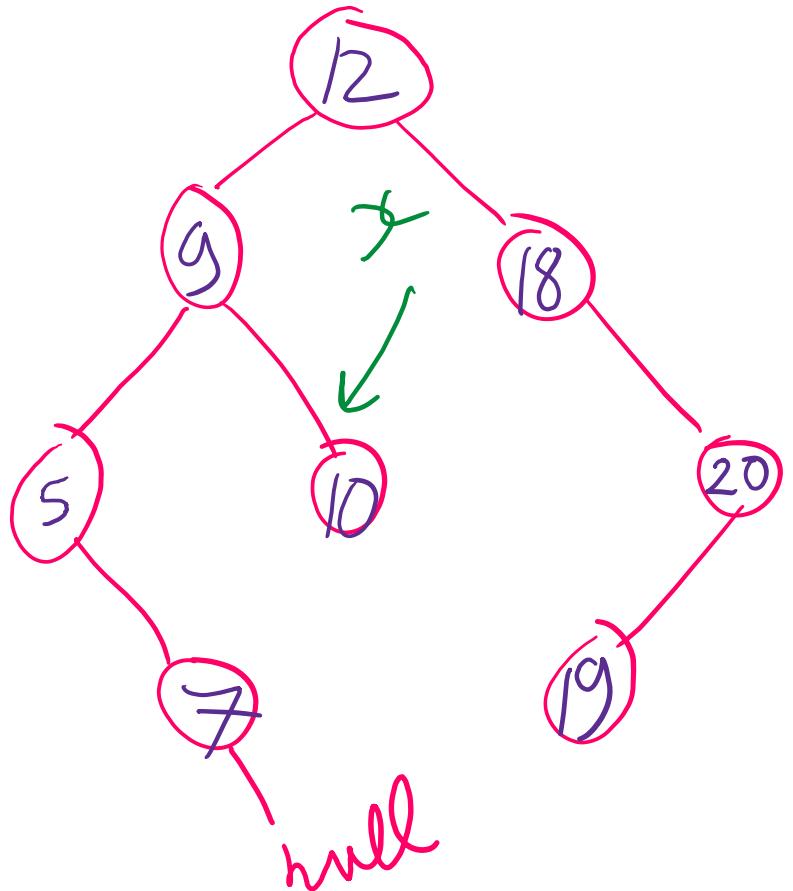
Iterative Inorder traversal



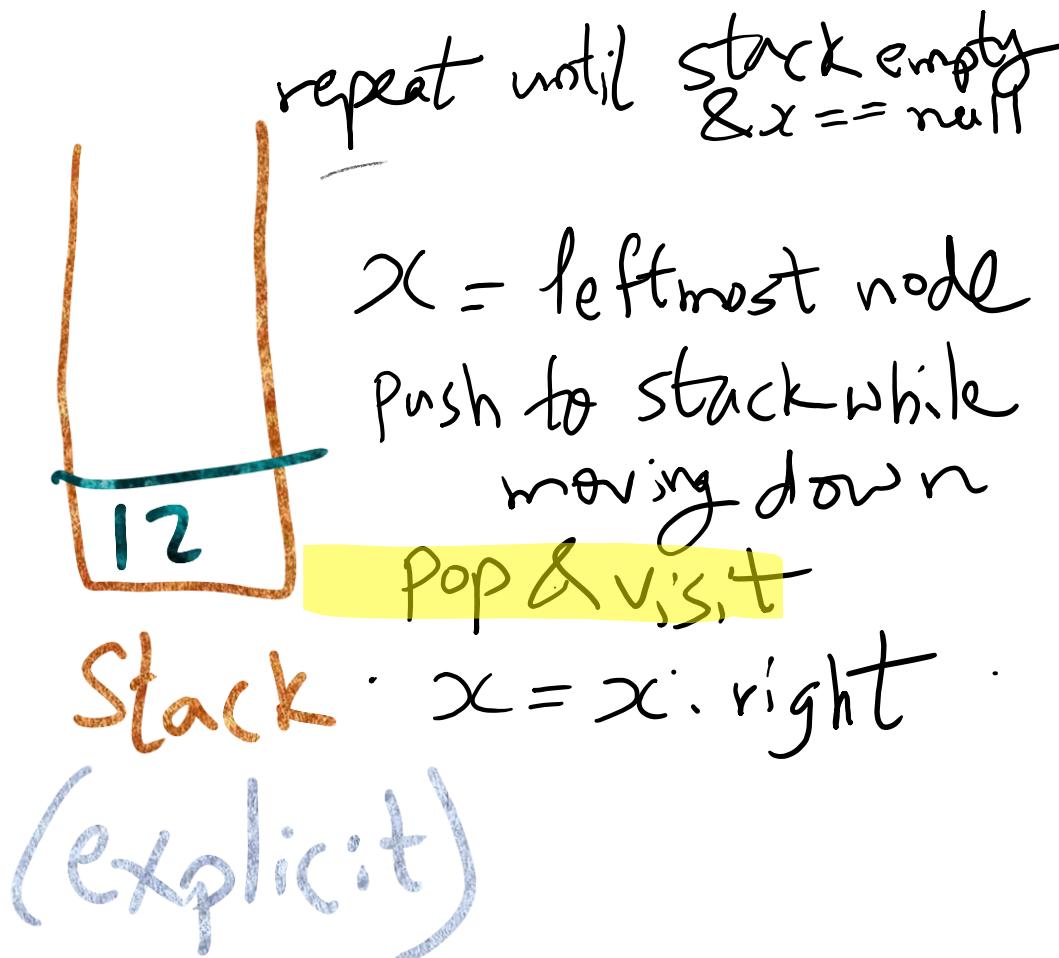
Visit order : 5, 7, 9



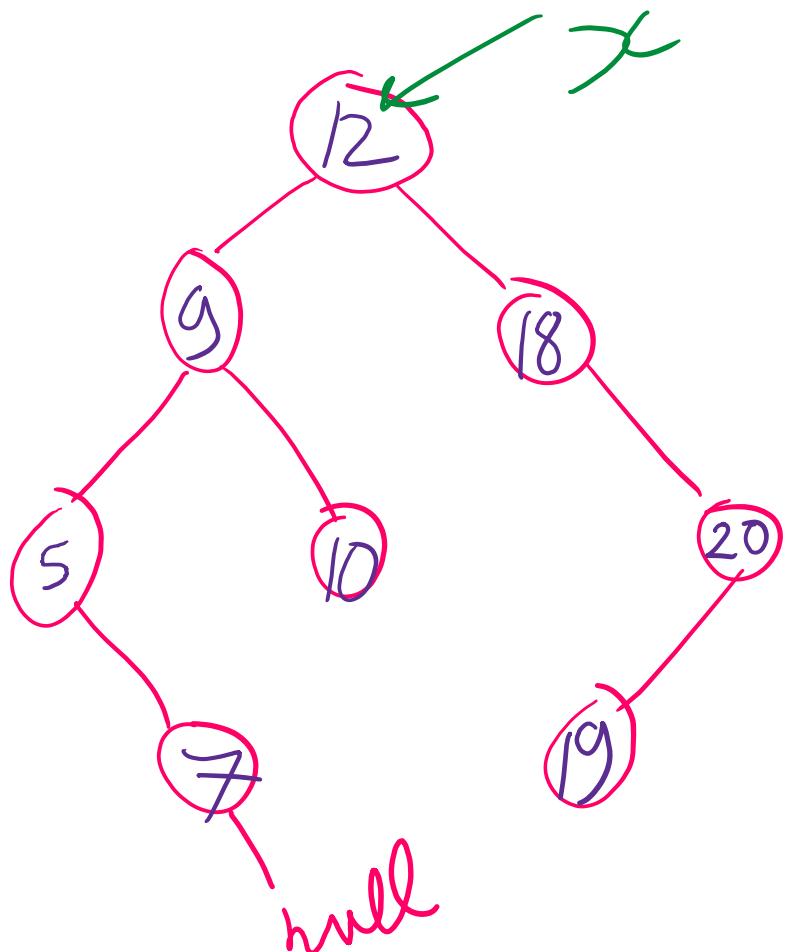
Iterative Inorder traversal



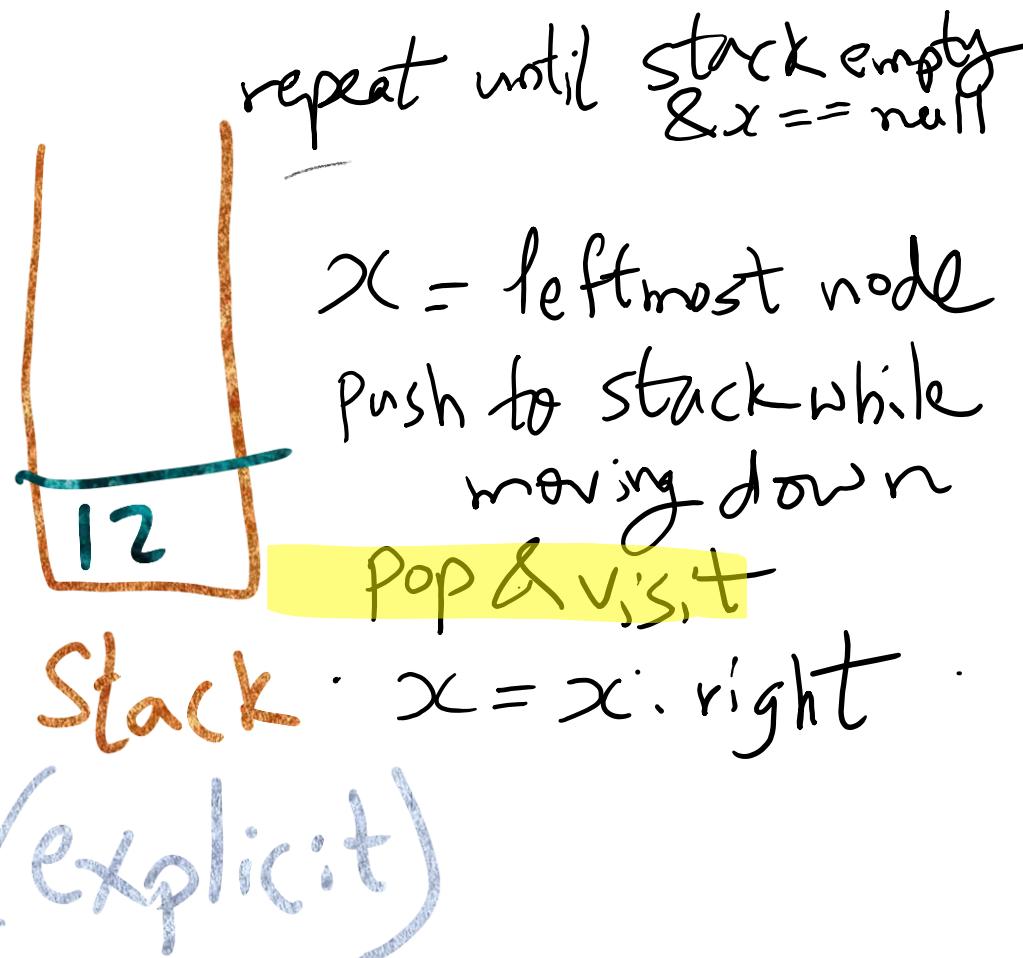
Visit order : 5, 7, 9, 10



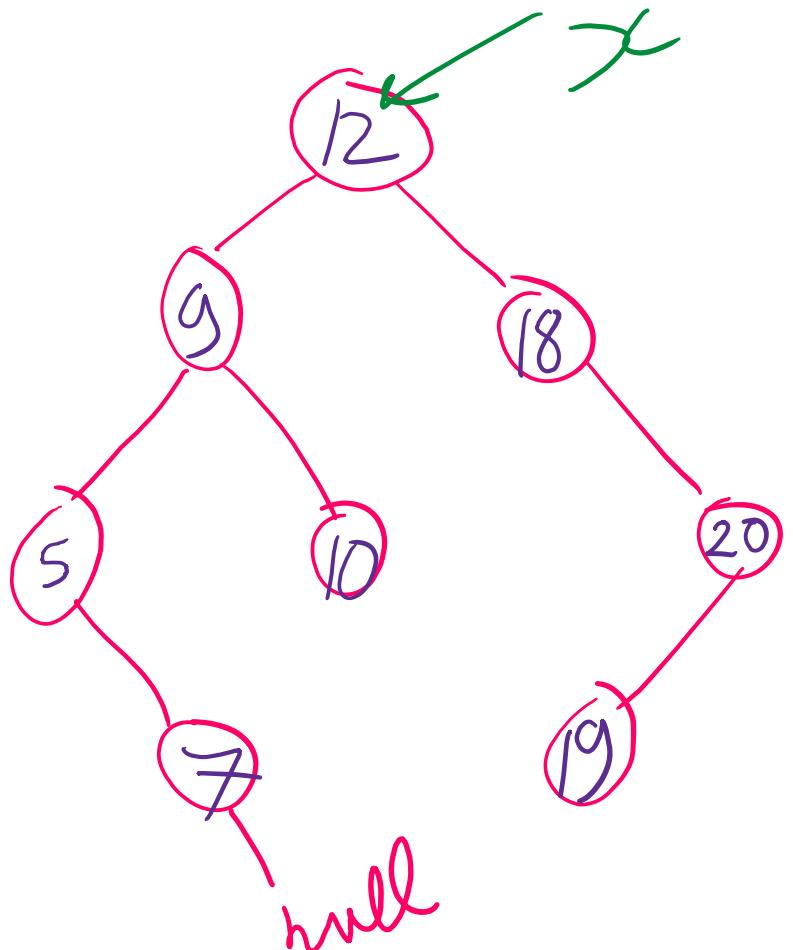
Iterative Inorder traversal



Visit order : 5, 7, 9, 10



Iterative Inorder traversal



Visit order : 5, 7, 9, 10, 12

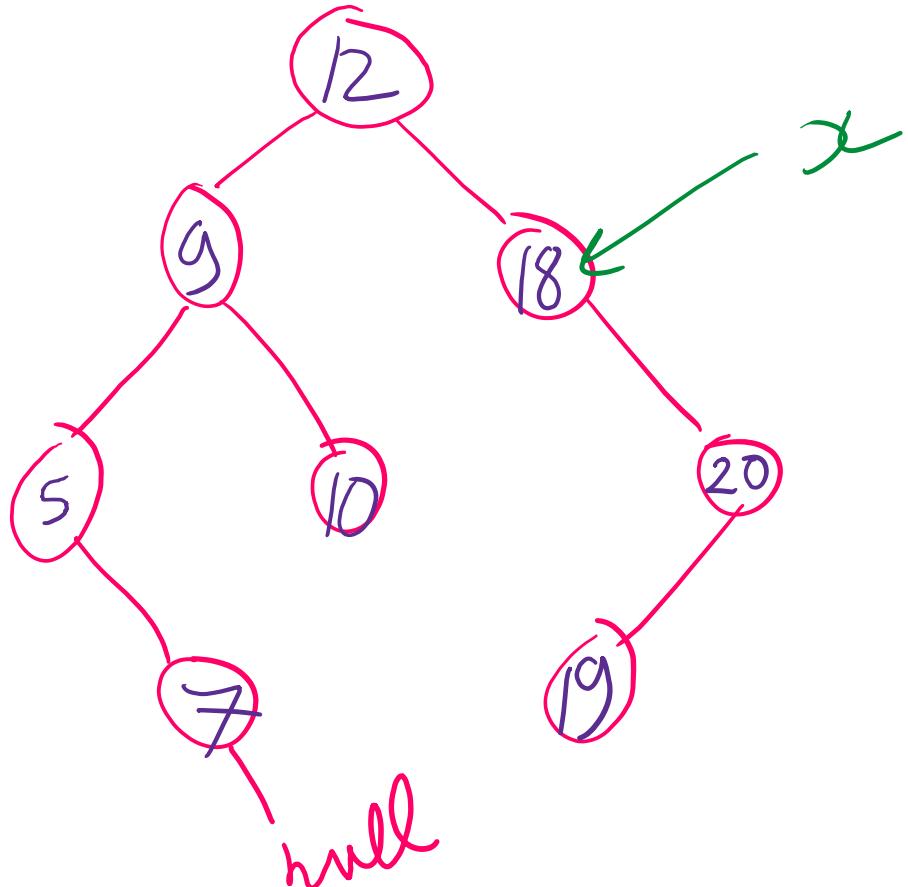
repeat until stack empty & $x == \text{null}$

$x = \text{leftmost node}$
push to stack while moving down

pop & visit

Stack : $x = x.\text{right}$
(explicit)

Iterative Inorder traversal



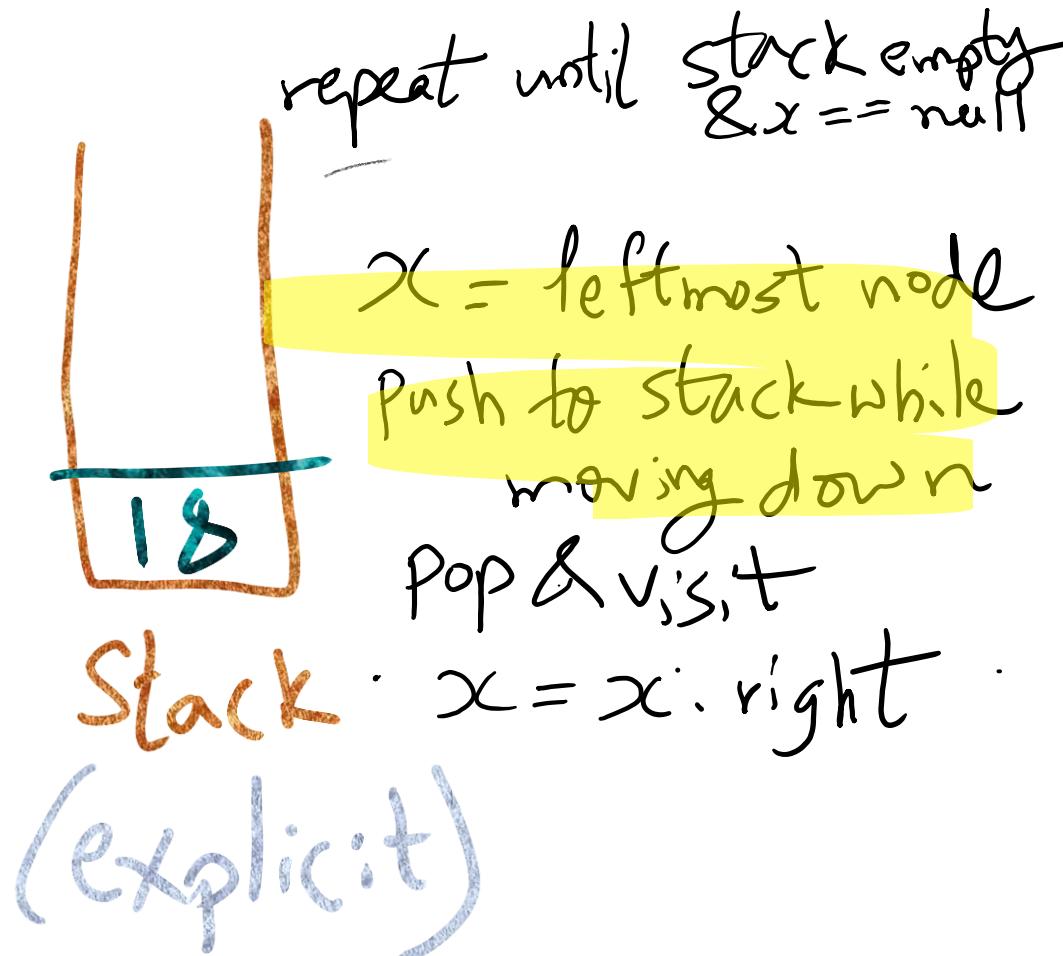
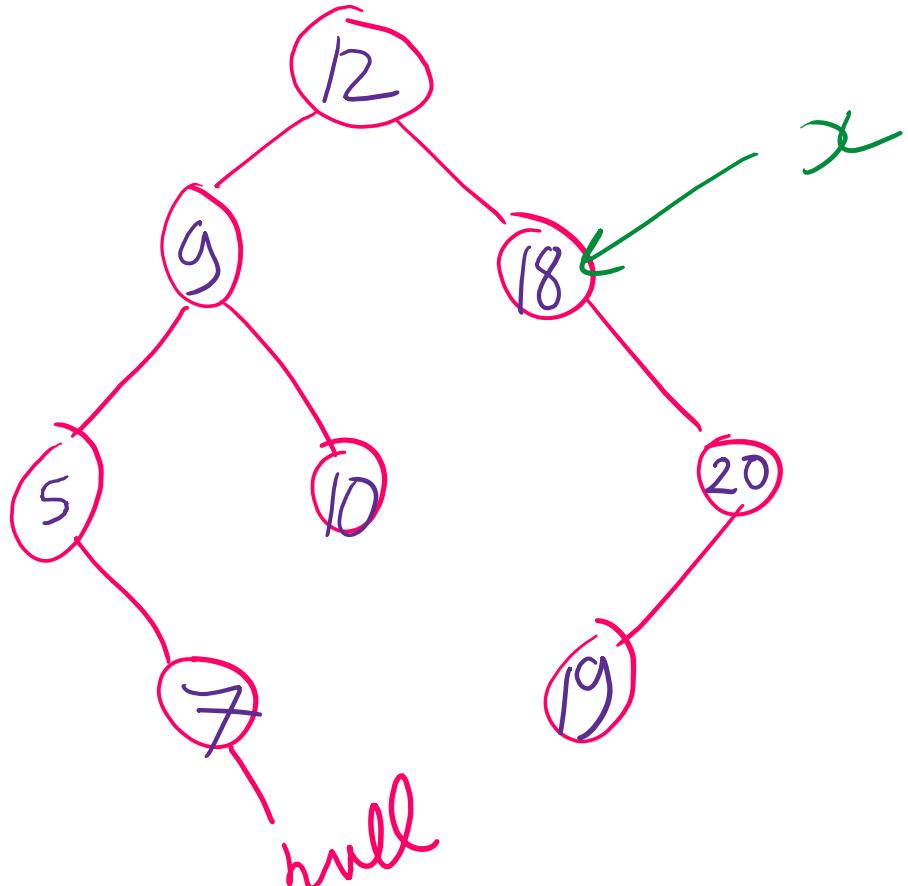
repeat until stack empty
& $x == \text{null}$

$x = \text{leftmost node}$
push to stack while moving down
pop & visit

Stack: $x = x.\text{right}$
(explicit)

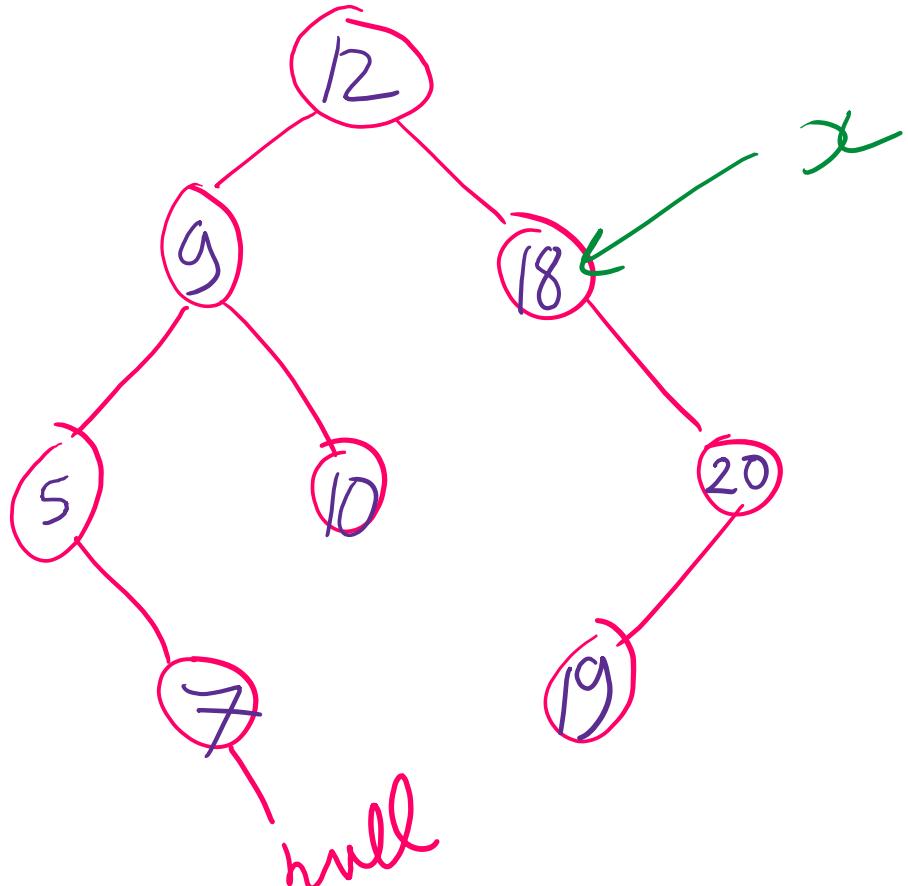
Visit order: 5, 7, 9, 10, 12

Iterative Inorder traversal



Visit order : 5, 7, 9, 10, 12

Iterative Inorder traversal



repeat until stack empty
 $\& x == \text{null}$

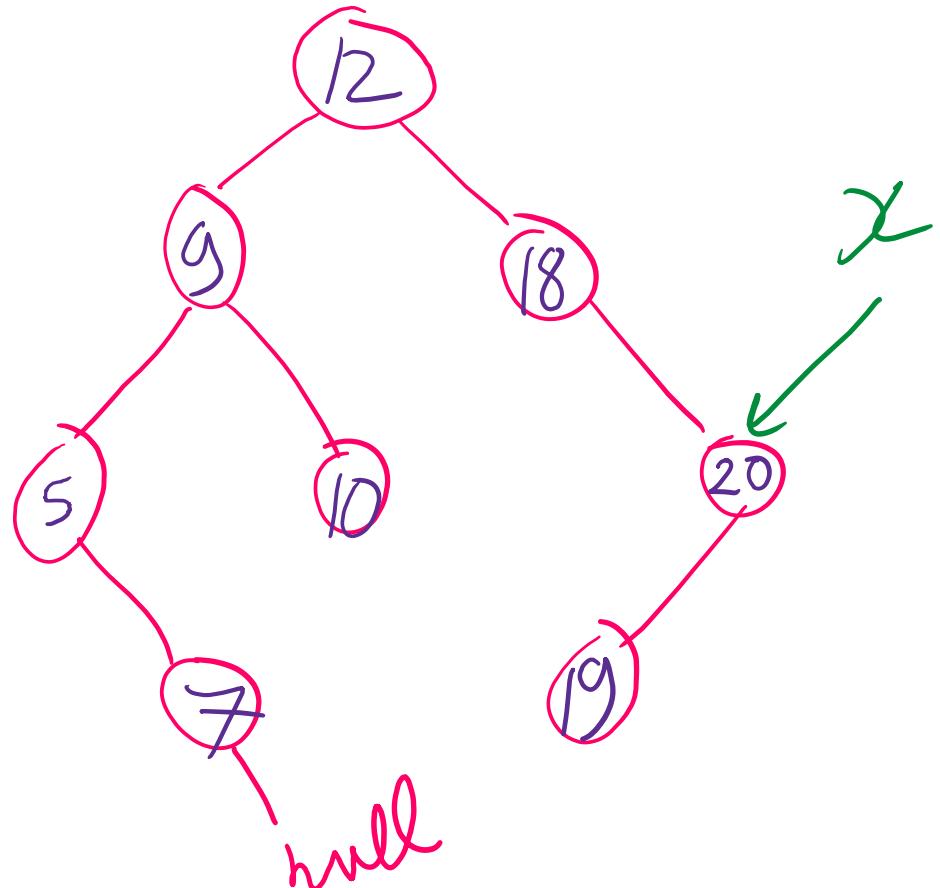
$x = \text{leftmost node}$
push to stack while moving down

pop & visit

Stack : $x = x.\text{right}$
(explicit)

Visit order : 5, 7, 9, 10, 12, 18

Iterative Inorder traversal



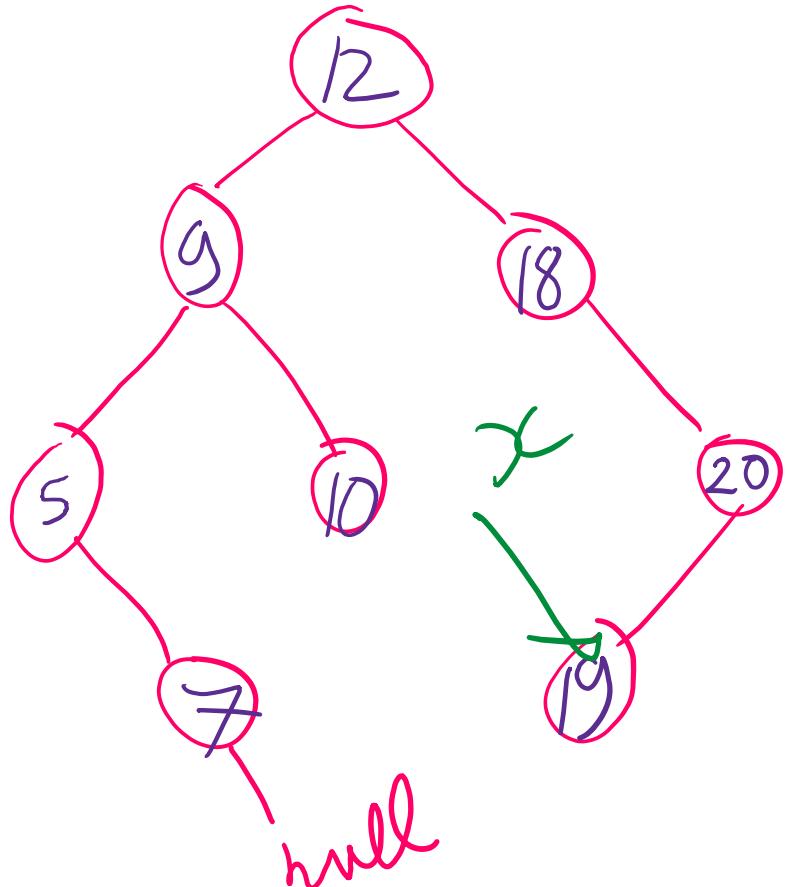
repeat until stack empty
& $x == \text{null}$

$x = \text{leftmost node}$
push to stack while moving down
pop & visit

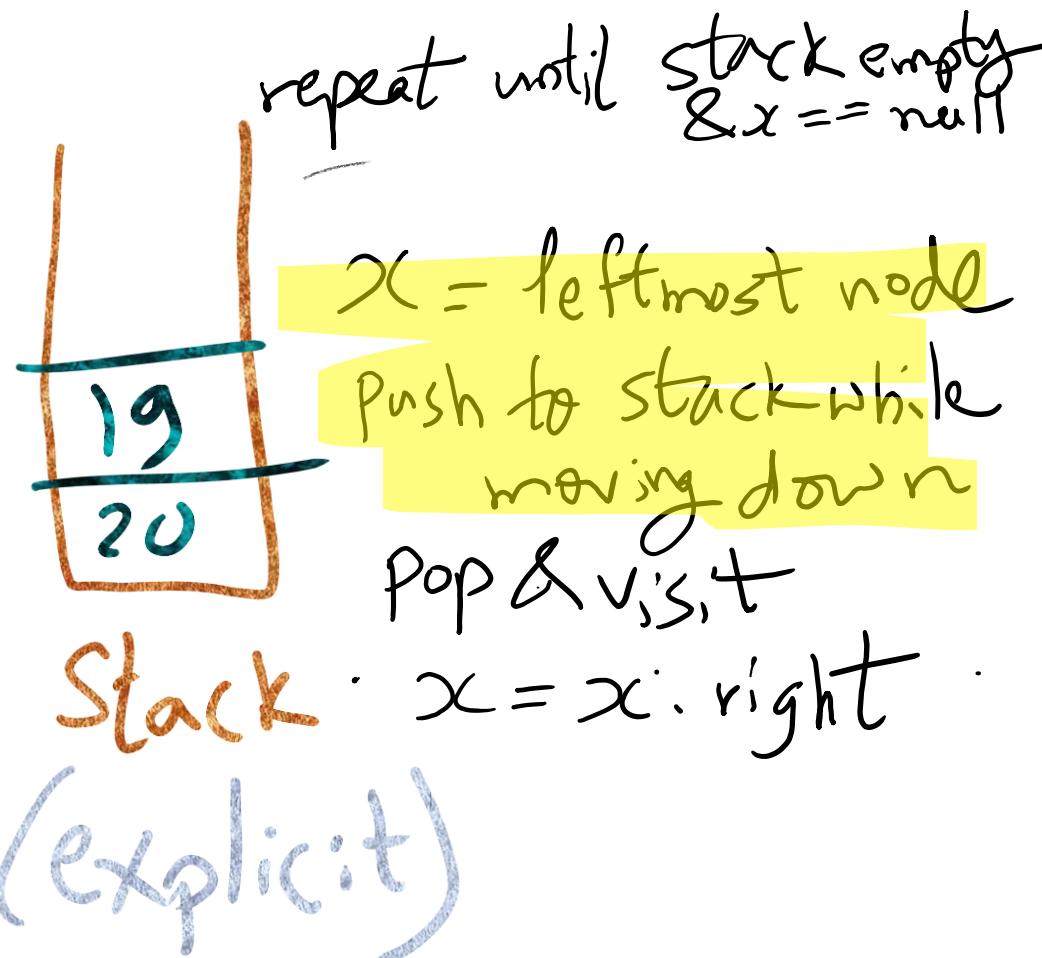
Stack : $x = x.\text{right}$
(explicit)

Visit order : 5, 7, 9, 10, 12, 18

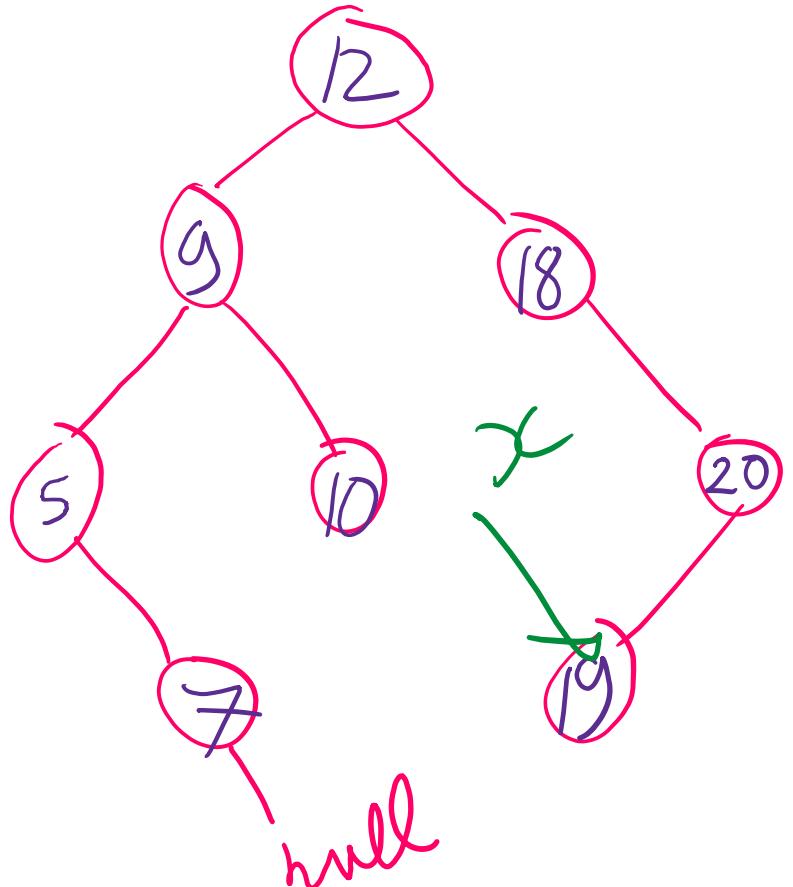
Iterative Inorder traversal



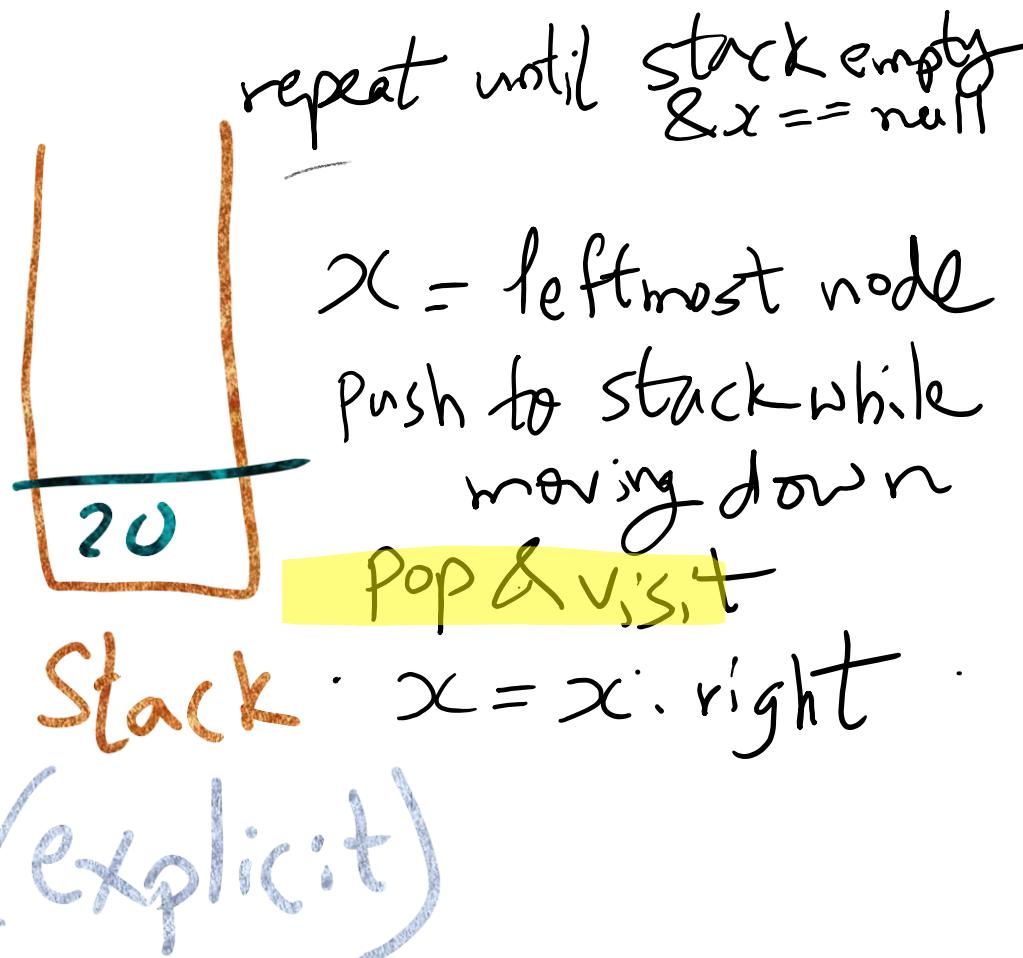
Visit order : 5, 7, 9, 10, 12, 18



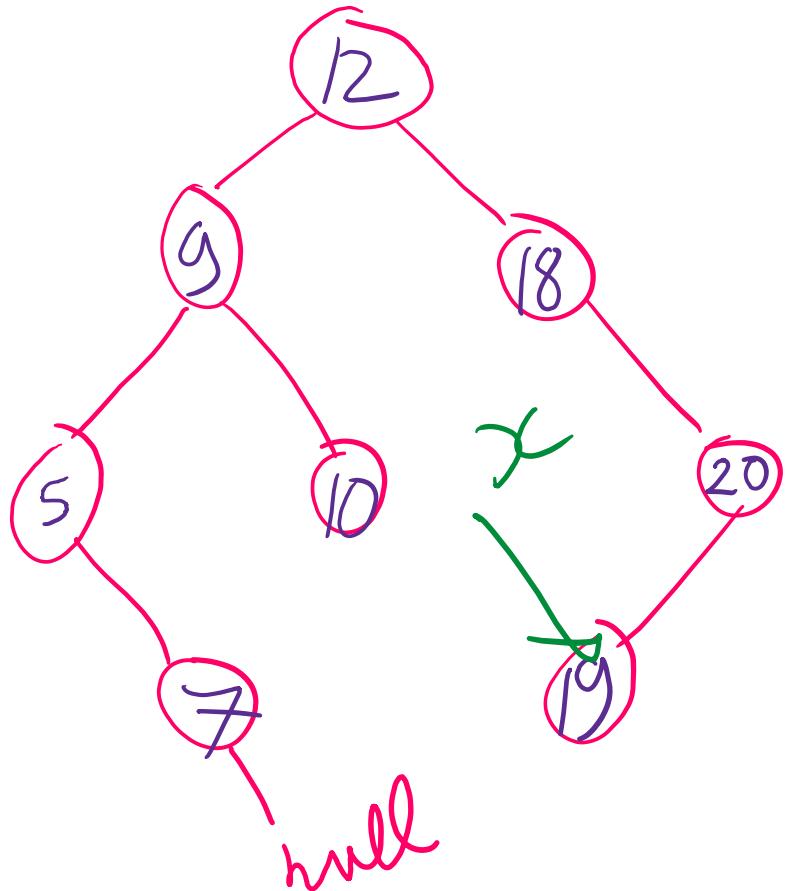
Iterative Inorder traversal



Visit order : 5, 7, 9, 10, 12, 18, 19



Iterative Inorder traversal



Visit order : 5, 7, 9, 10, 12, 18, 19

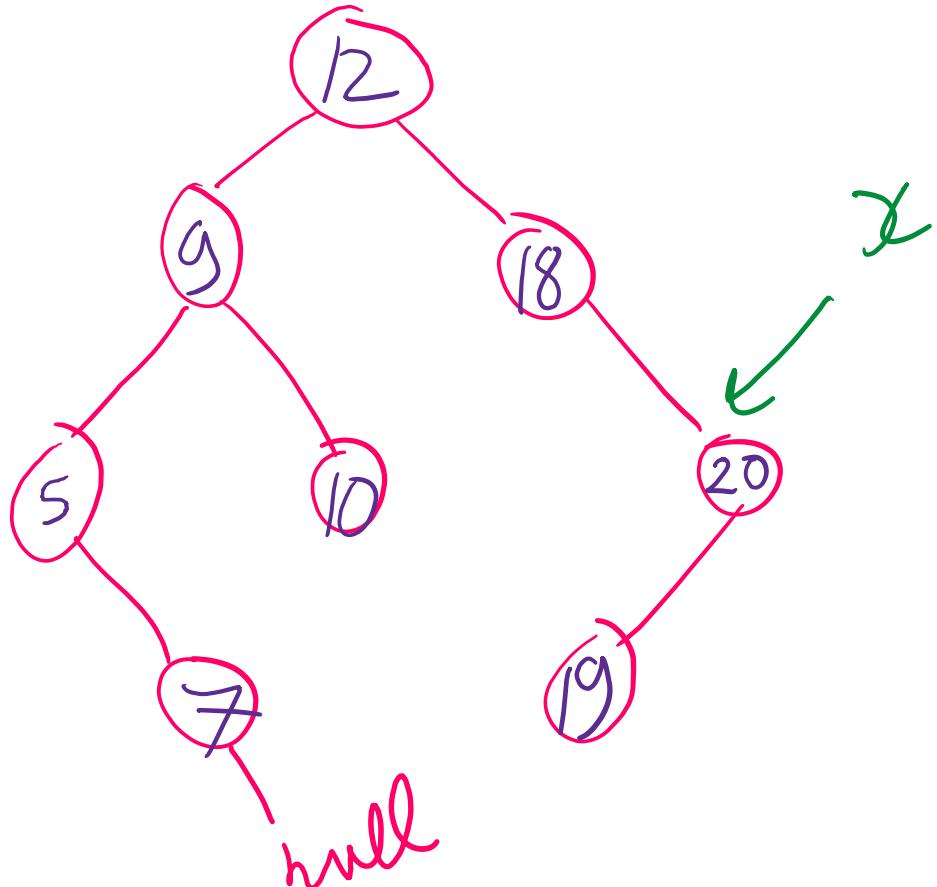
repeat until stack empty
 $\& x == \text{null}$



$x = \text{leftmost node}$
push to stack while moving down
pop & visit
 $x = x.\text{right}$

Stack
(explicit)

Iterative Inorder traversal



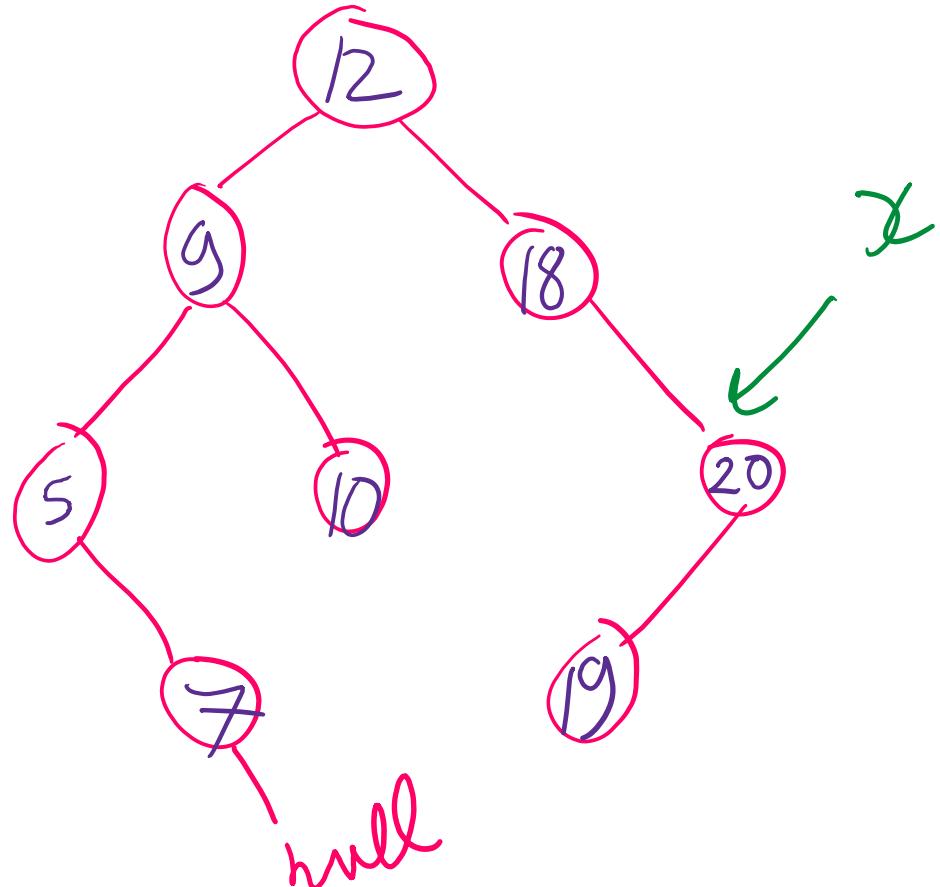
repeat until stack empty
 $\& x == \text{null}$

$x = \text{leftmost node}$
push to stack while moving down
pop & visit

Stack : $x = x.\text{right}$
(explicit)

Visit order : 5, 7, 9, 10, 12, 18, 19, 20

Iterative Inorder traversal



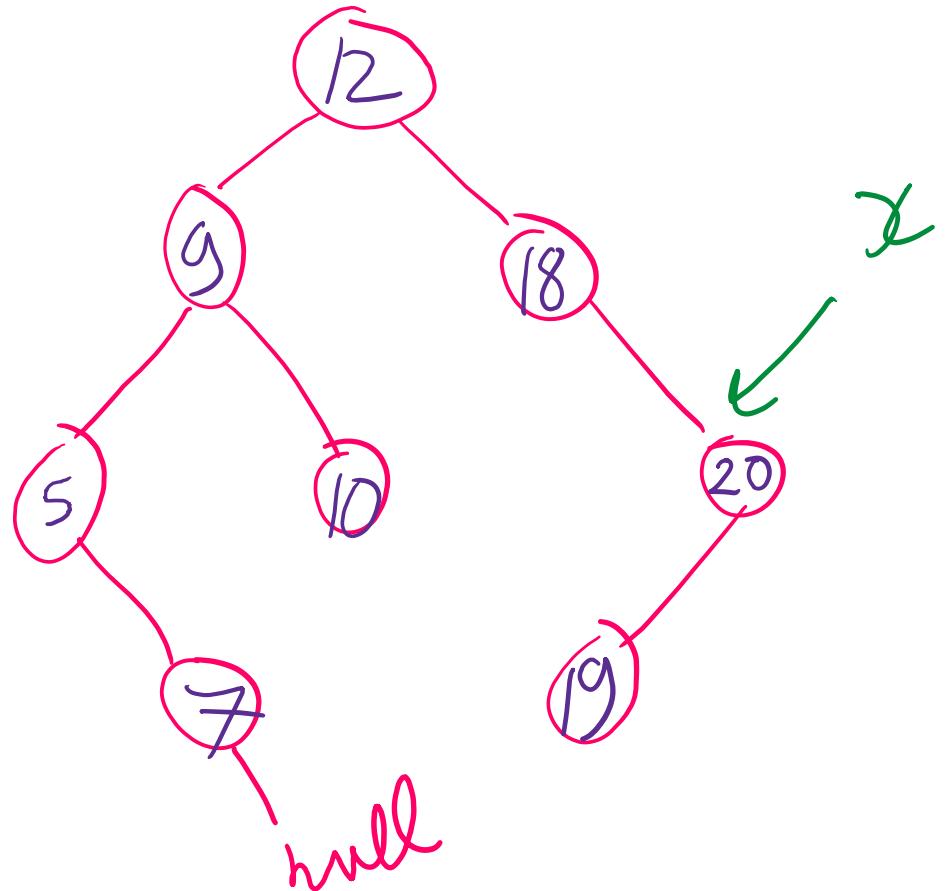
repeat until stack empty
 $\& x == \text{null}$

$x = \text{leftmost node}$
push to stack while moving down
pop & visit

Stack : $x = x.\text{right}$
(explicit)

Visit order : 5, 7, 9, 10, 12, 18, 19, 20

Iterative Inorder traversal



Visit order : 5, 7, 9, 10, 12, 18, 19, 20

repeat until stack empty & $x == \text{null}$

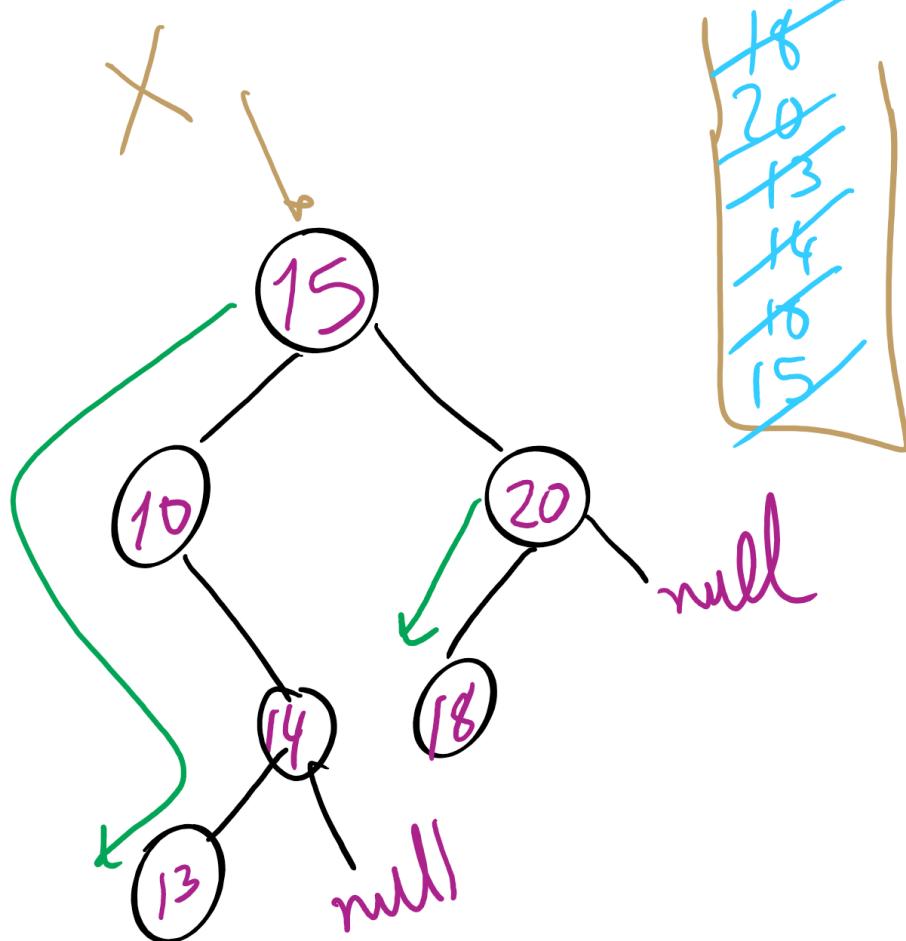
$x = \text{leftmost node}$
push to stack while moving down
pop & visit
 $x = x.\text{right}$

Stack (explicit)

Traversals of a Binary Tree

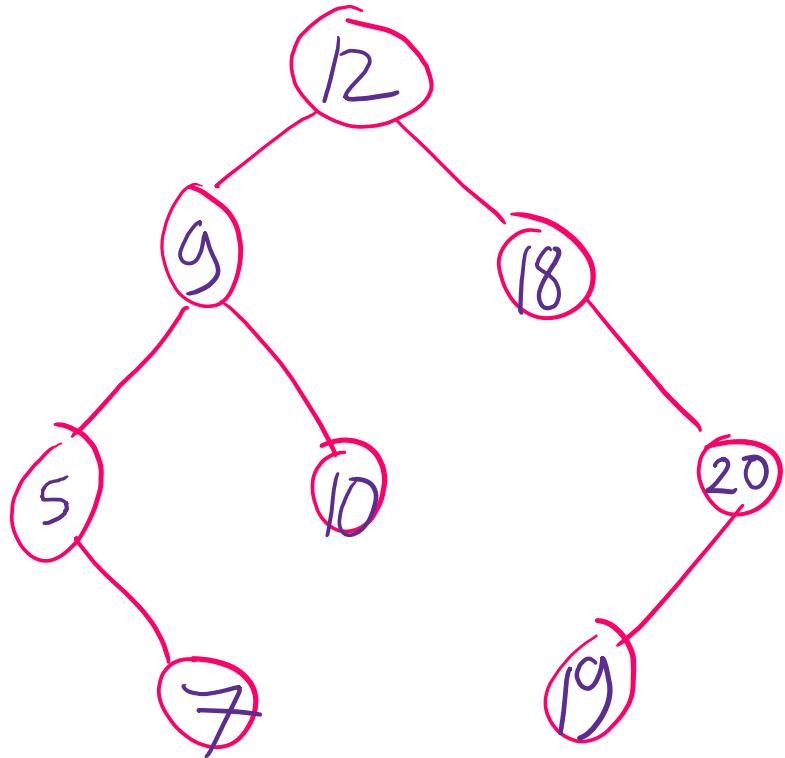
- Preorder traversal
 - Visit root before we visit root's subtrees
- Inorder traversal
 - Visit root of a binary tree between visiting nodes in root's subtrees.
- Postorder traversal
 - Visit root of a binary tree after visiting nodes in root's subtrees
 - left then right then root

Iterative Post-order Traversal

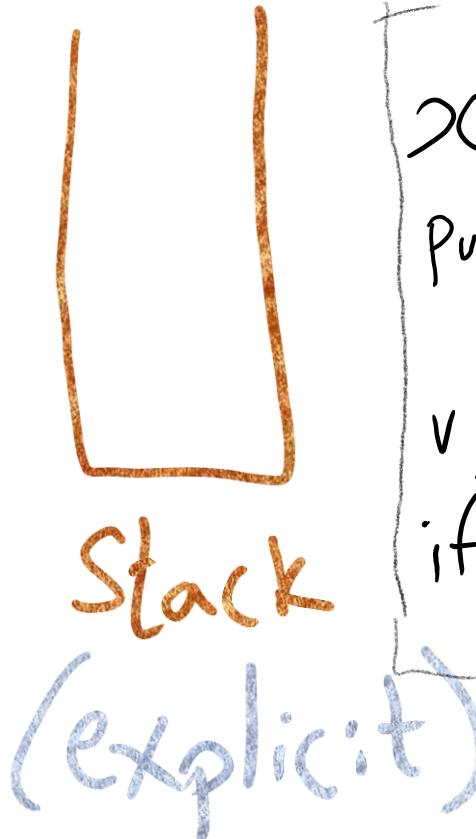


13, 14, 10, 18, 20 15

Iterative Postorder traversal



Visit order :

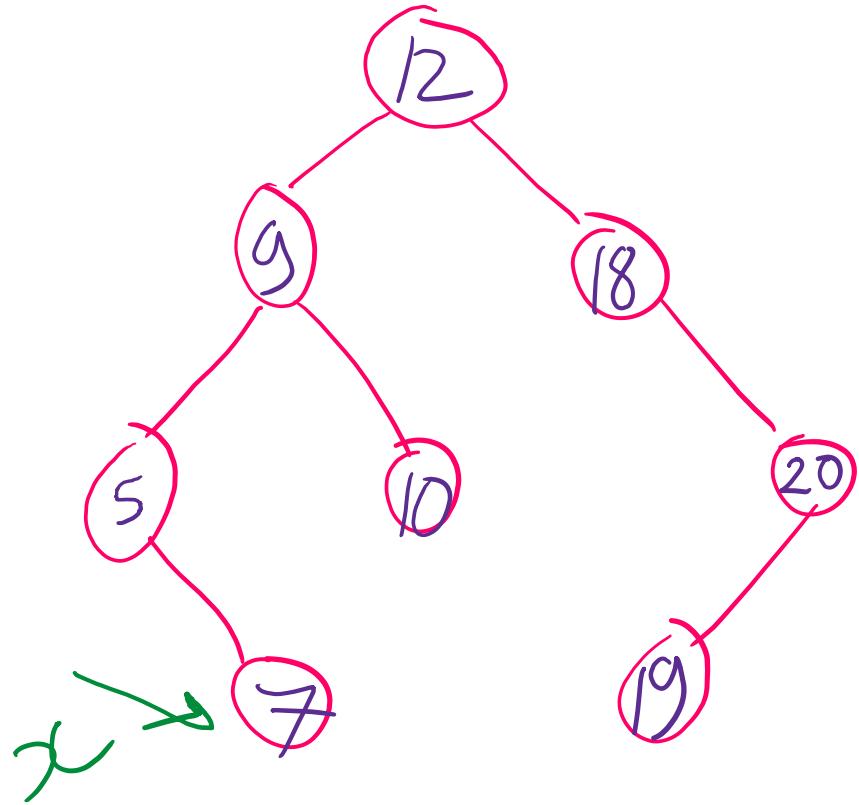


repeat until stack empty
 $\& x == \text{null}$

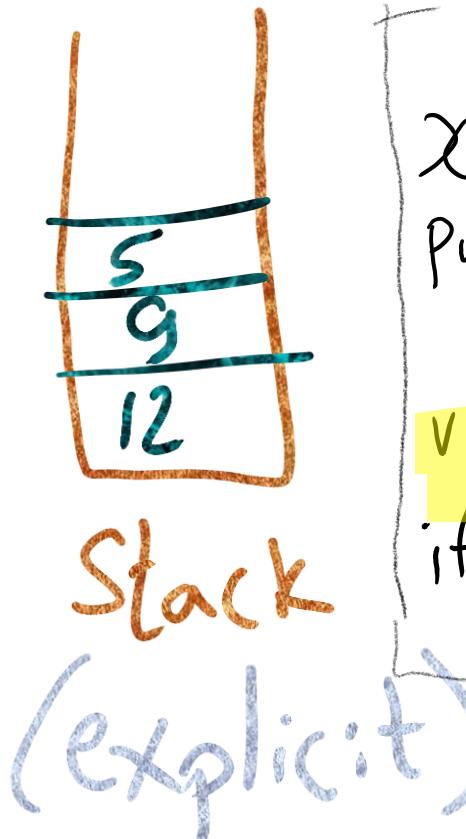
$x = \text{leftmost leaf}$
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf

else $x = \text{parent, right}$

Iterative Postorder traversal



Visit order : 7

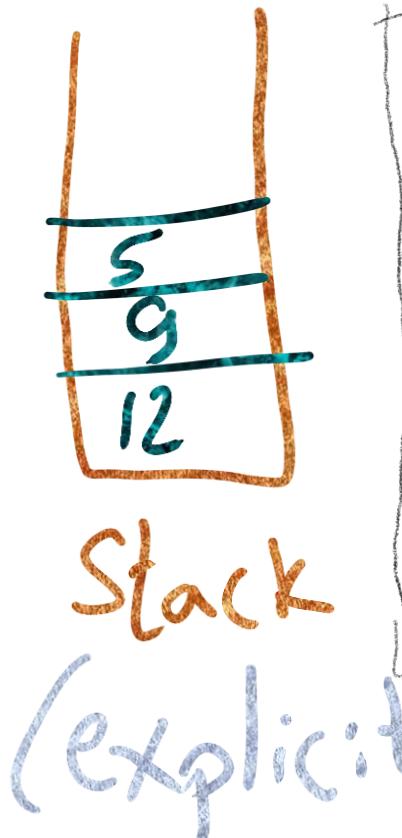
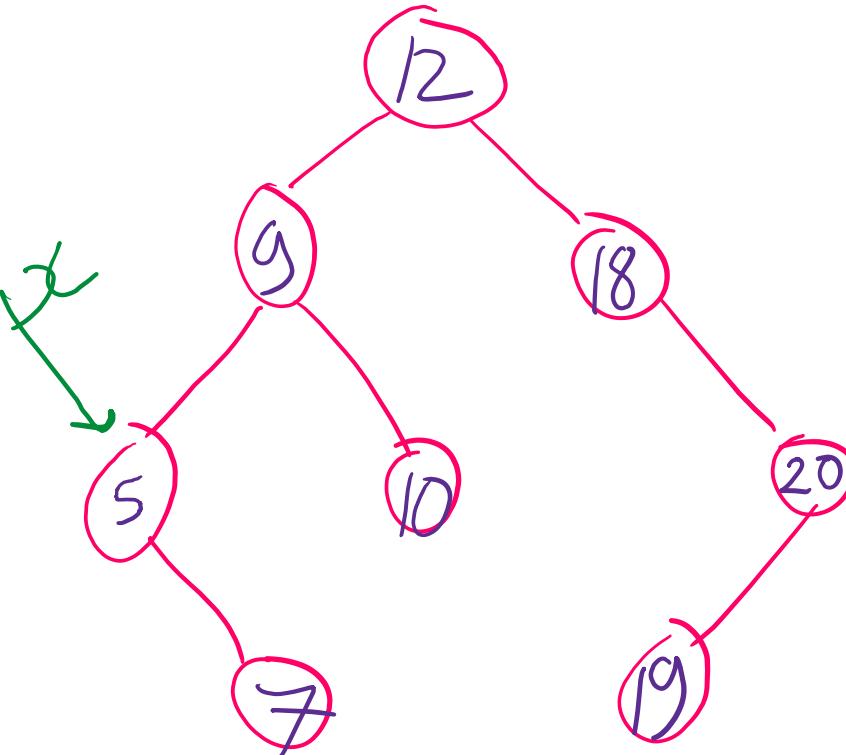


repeat until stack empty & $x = \text{null}$

$x = \text{leftmost leaf}$
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf

else $x = \text{parent. right}$

Iterative Postorder traversal



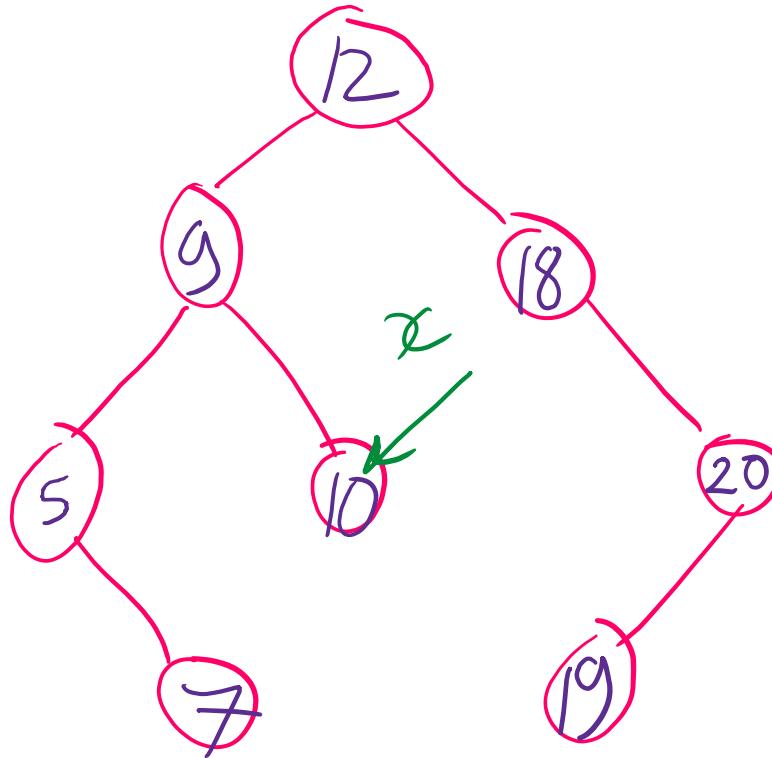
repeat until stack empty & $x = \text{null}$

$x = \text{leftmost leaf}$
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf

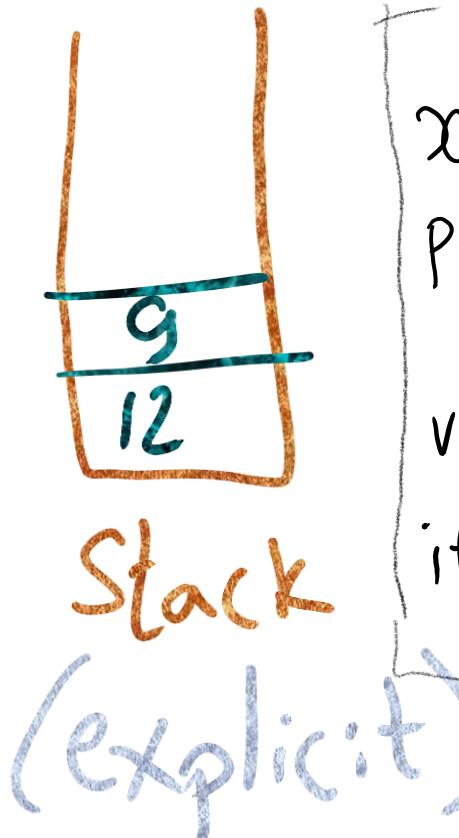
Visit order : 7, 5

else $x = \text{parent. right}$

Iterative Postorder traversal



Visit order : 7, 5

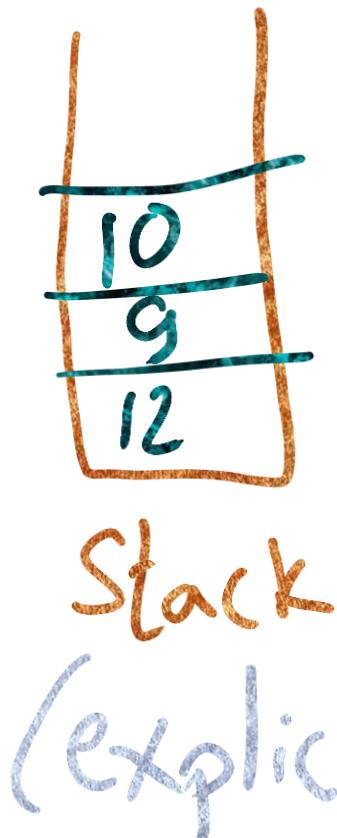
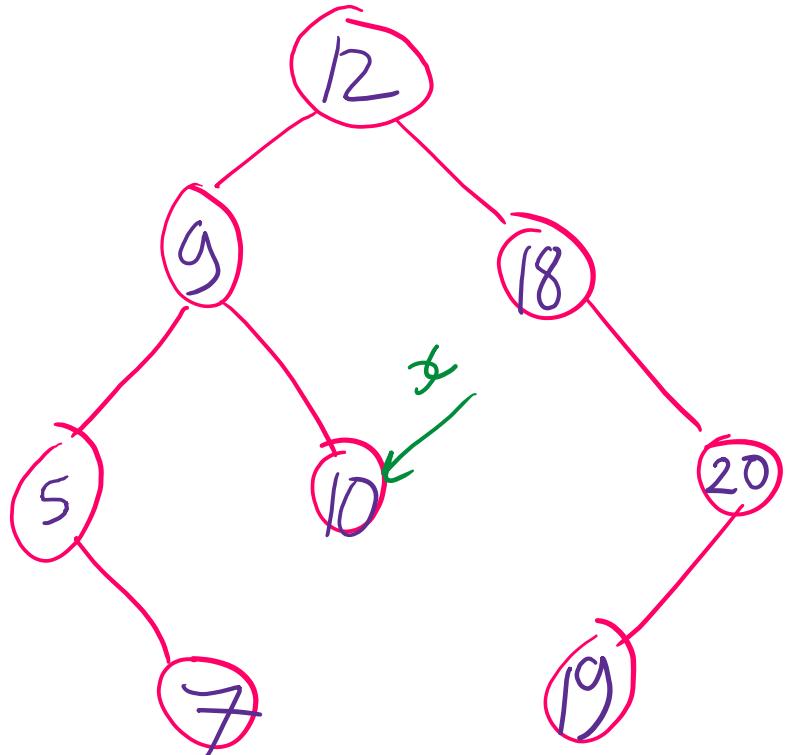


repeat until stack empty & $x = \text{null}$

$x = \text{leftmost leaf}$
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf

else $x = \text{parent. right}$

Iterative Postorder traversal



repeat until stack empty & $x = \text{null}$

$x = \text{leftmost leaf}$

Push to stack while moving down

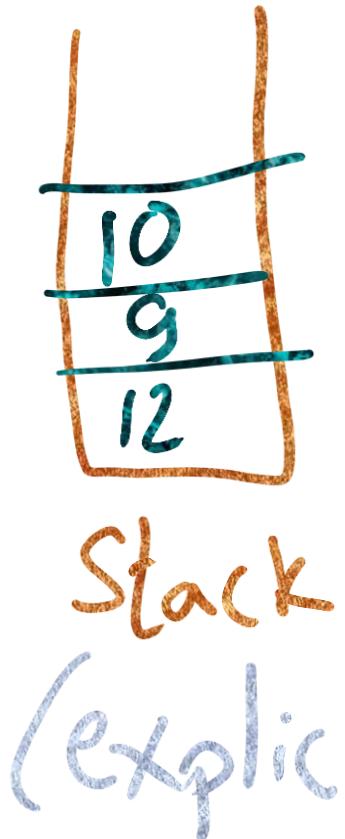
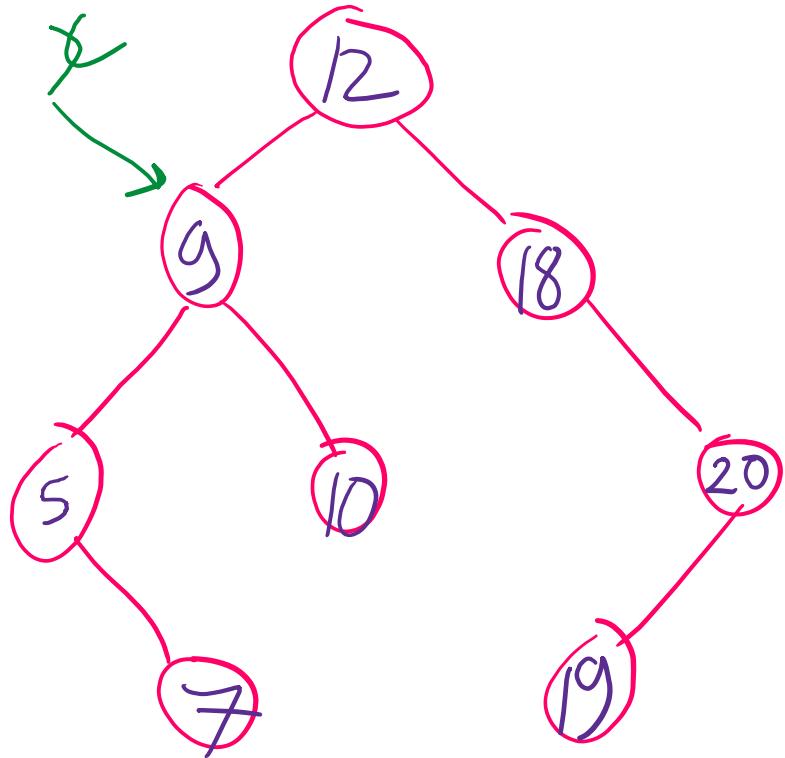
visit x

if x is right child
pop & visit
skip leftmost leaf

Visit order: 7, 5, 10

else $x = \text{parent. right}$

Iterative Postorder traversal



repeat until stack empty & $x = \text{null}$

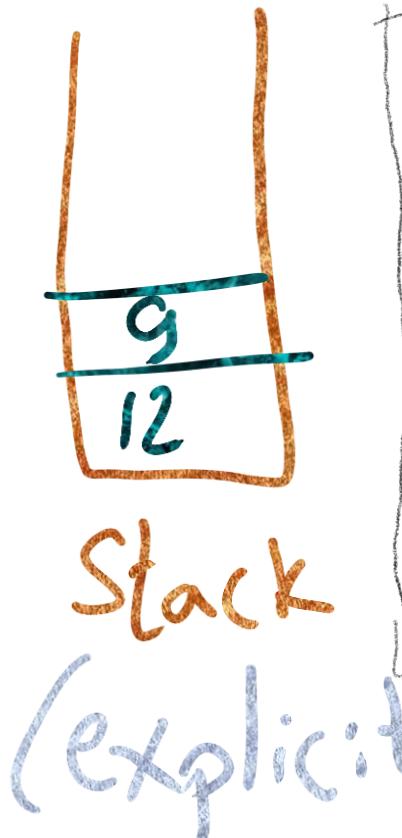
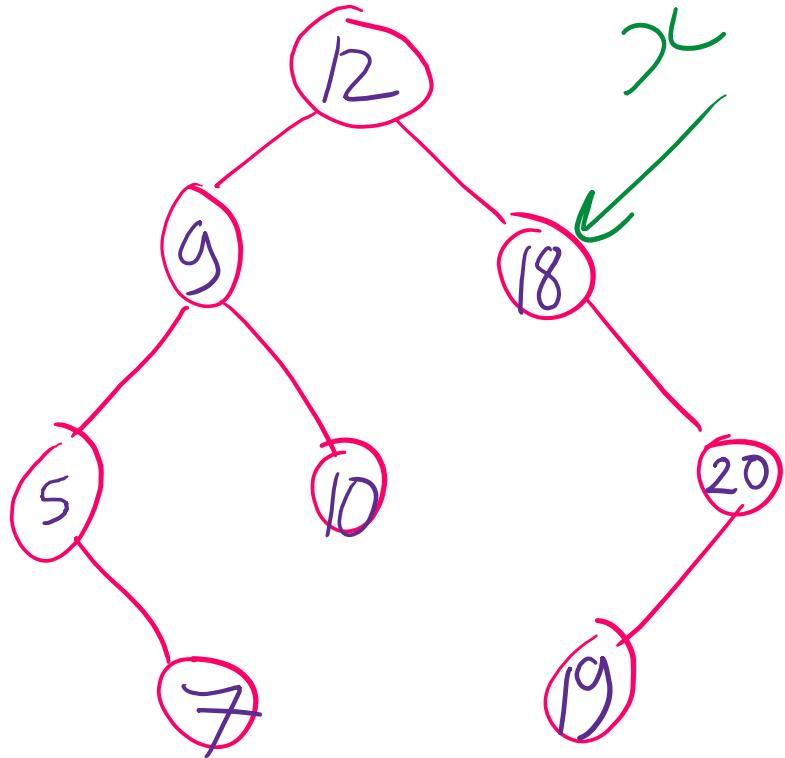
$x = \text{leftmost leaf}$
push to stack while moving down
visit x

if x is rightchild
pop & visit
skip leftmost leaf

Visit order: 7, 5, 10, 9

else $x = \text{parent. right}$

Iterative Postorder traversal



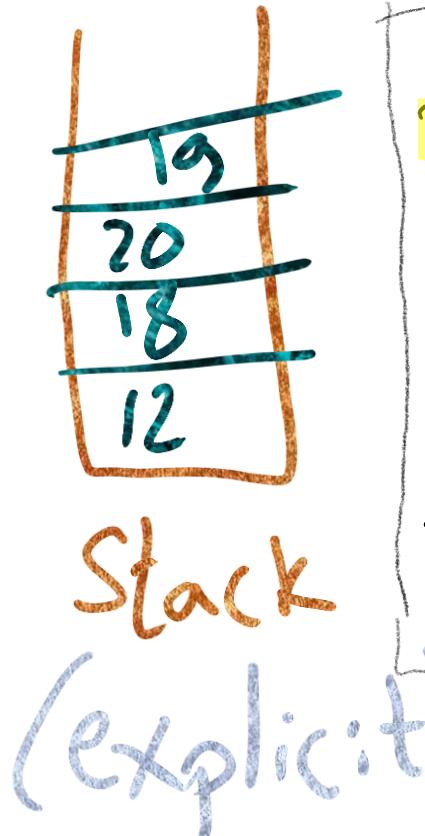
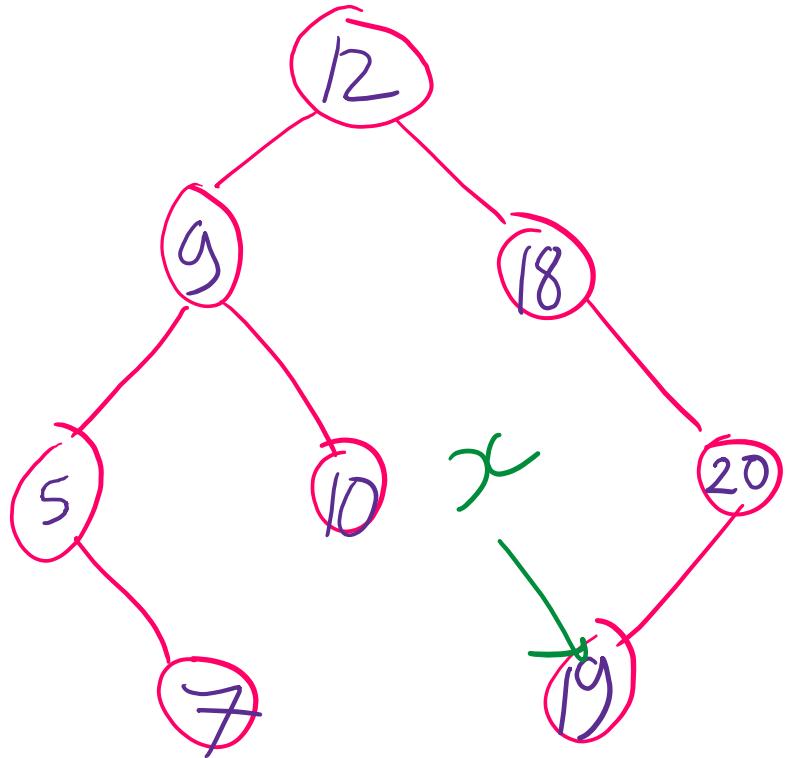
repeat until stack empty & $x = \text{null}$

$x = \text{leftmost leaf}$
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf

Visit order: 7, 5, 10, 9

else $x = \text{parent. right}$

Iterative Postorder traversal



repeat until stack empty
& $x = \text{null}$

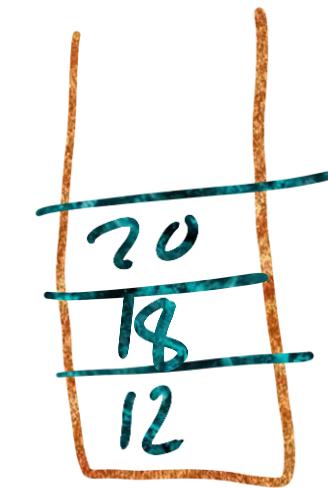
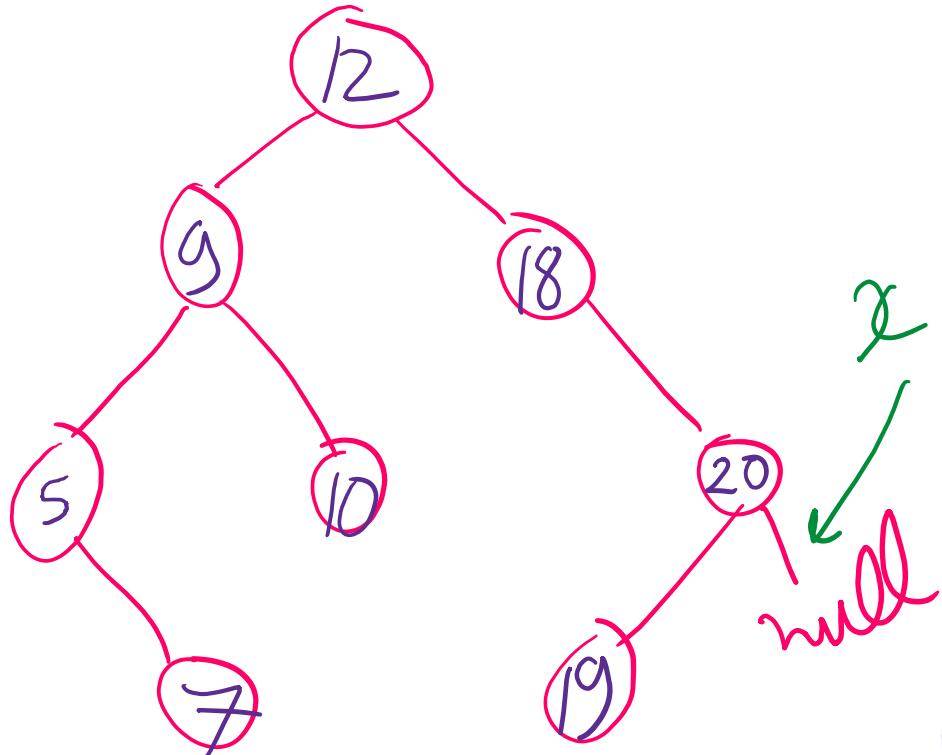
$x = \text{leftmost leaf}$
push to stack while moving down
visit x

if x is right child
pop & visit
skip leftmost leaf

else $x = \text{parent. right}$

Visit order: 7, 5, 10, 9, 19

Iterative Postorder traversal



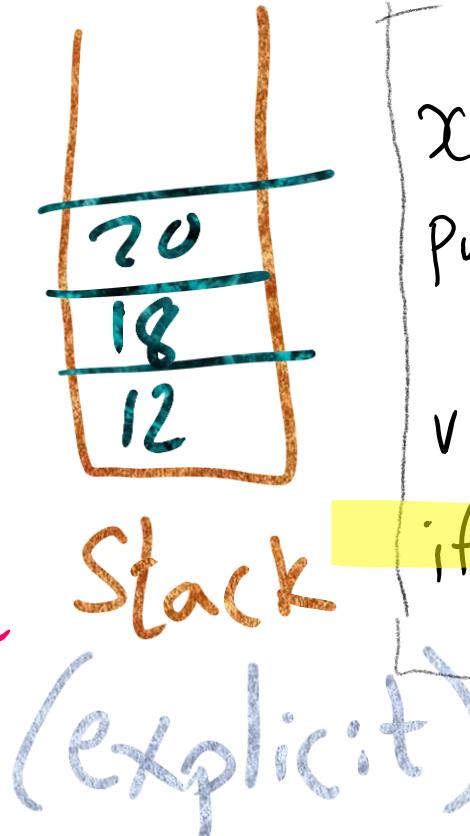
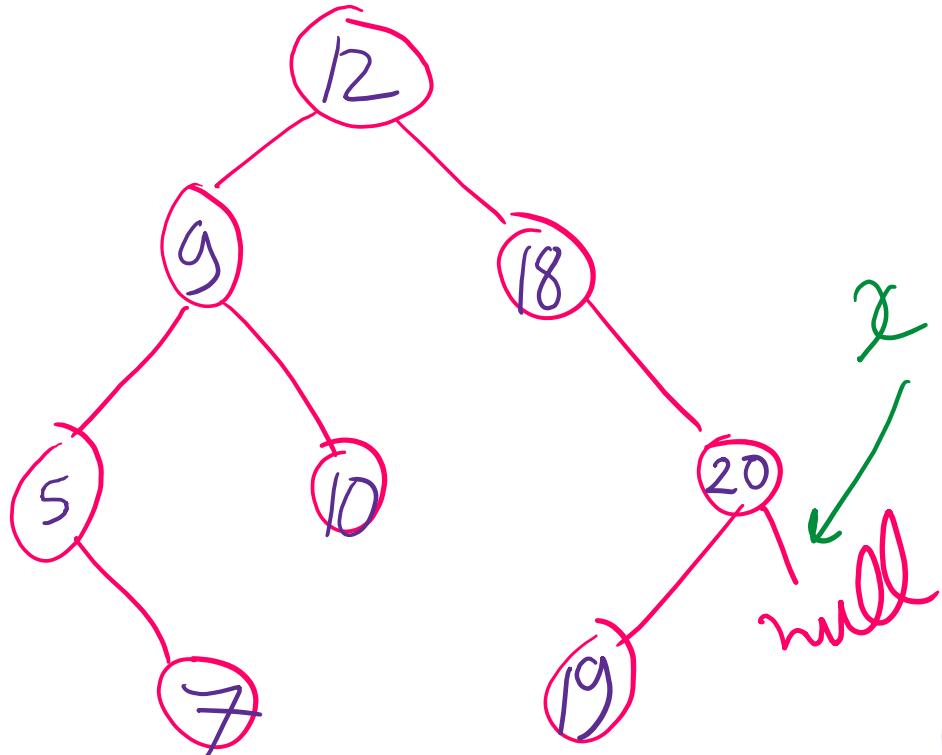
repeat until stack empty & $x == \text{null}$

$x = \text{leftmost leaf}$
push to stack while moving down
visit x
if x is right child
pop & visit
skip leftmost leaf

Visit order: 7, 5, 10, 9, 19

else $x = \text{parent right}$

Iterative Postorder traversal



repeat until stack empty
 $\& x == \text{null}$

$x = \text{leftmost leaf}$
push to stack while moving down

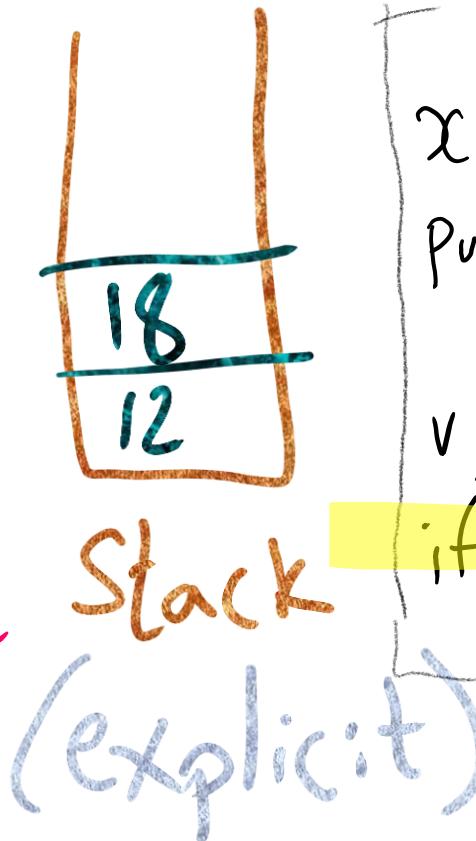
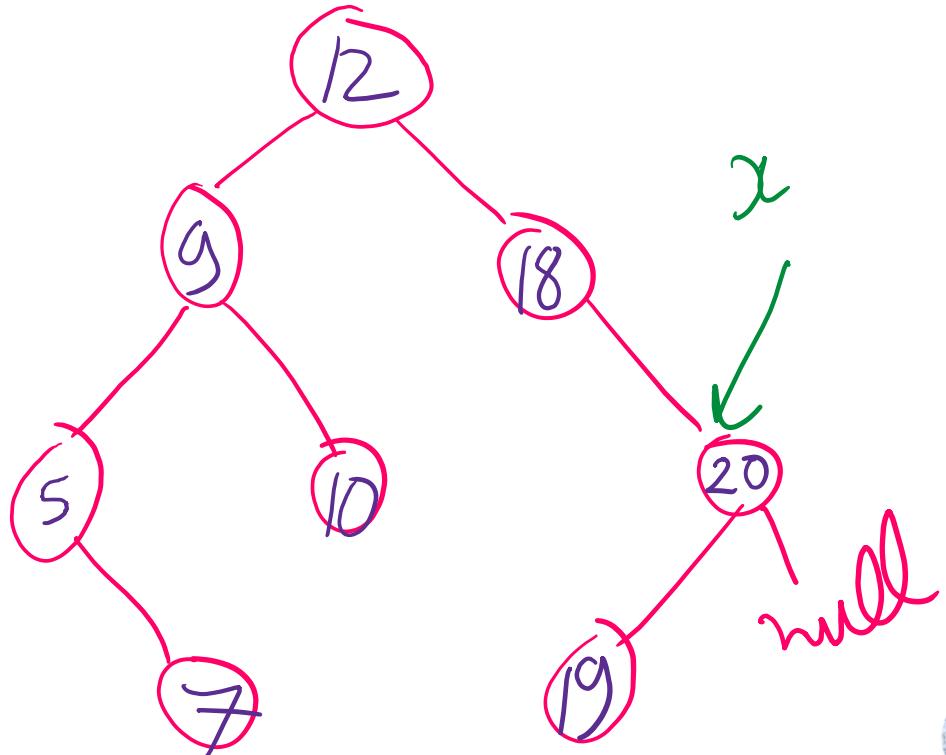
visit x

if x is right child
pop & visit
skip leftmost leaf

Visit order: 7, 5, 10, 9, 19

else $x = \text{parent. right}$

Iterative Postorder traversal



repeat until stack empty
 $\& x == \text{null}$

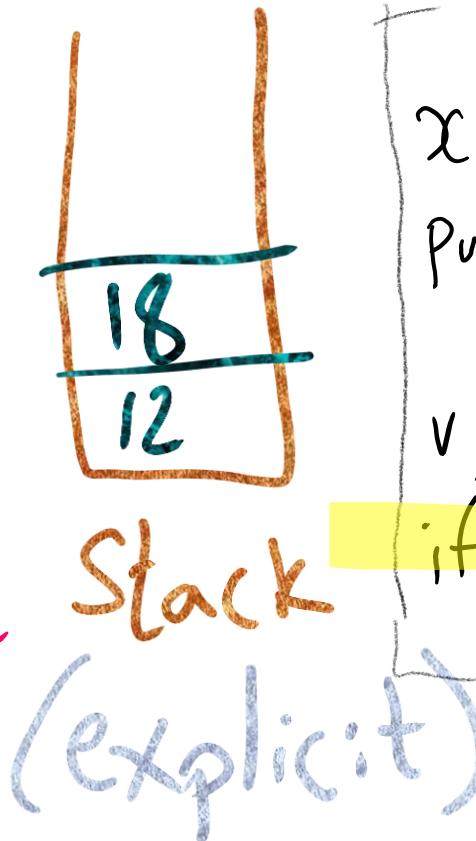
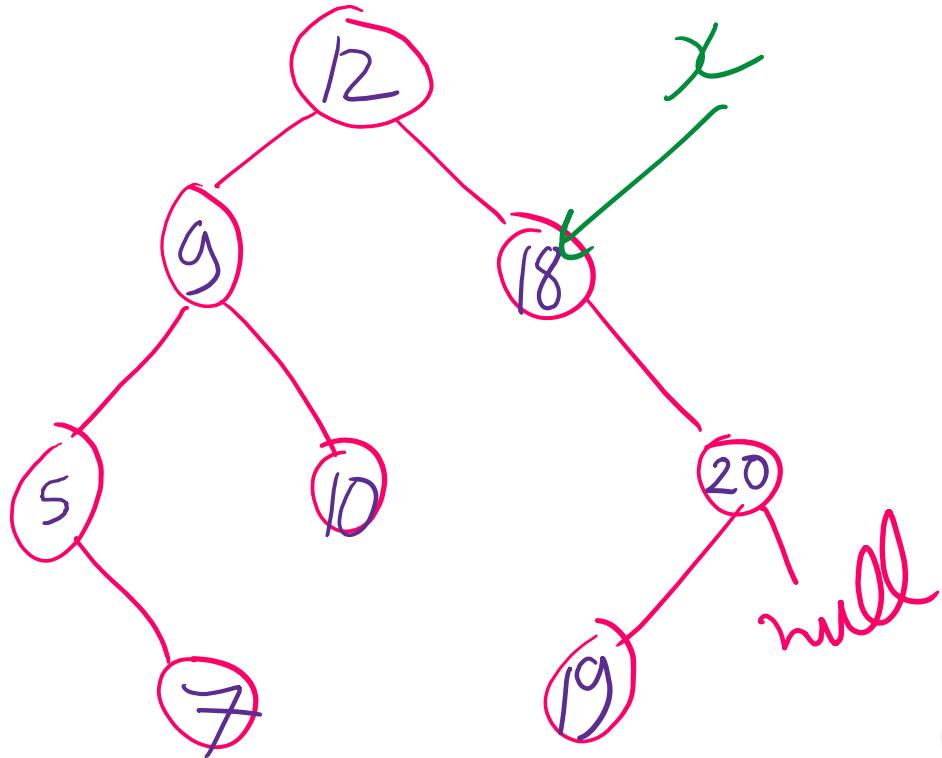
$x = \text{leftmost leaf}$
push to stack while moving down
visit x

if x is right child
pop & visit
skip leftmost leaf

Visit order: 7, 5, 10, 9, 19, 20

else $x = \text{parent. right}$

Iterative Postorder traversal



repeat until stack empty & $x = \text{null}$

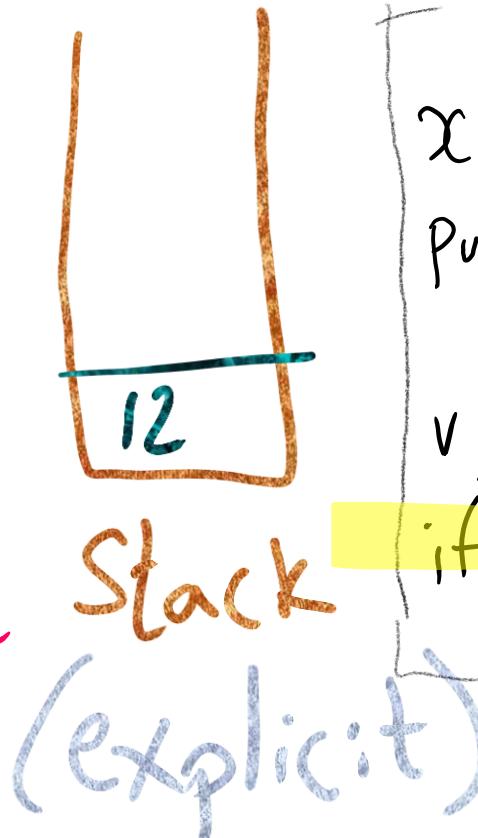
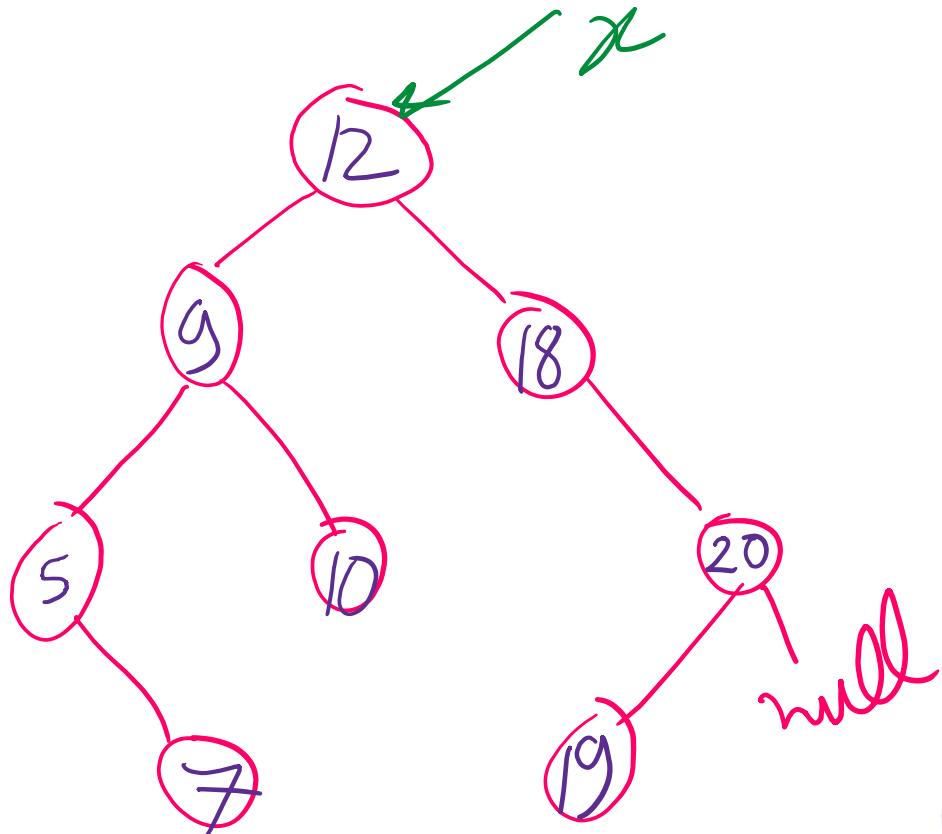
$x = \text{leftmost leaf}$
push to stack while moving down
visit x

if x is right child
pop & visit
skip leftmost leaf

Visit order: 7, 5, 10, 9, 19, 20, 18

else $x = \text{parent. right}$

Iterative Postorder traversal



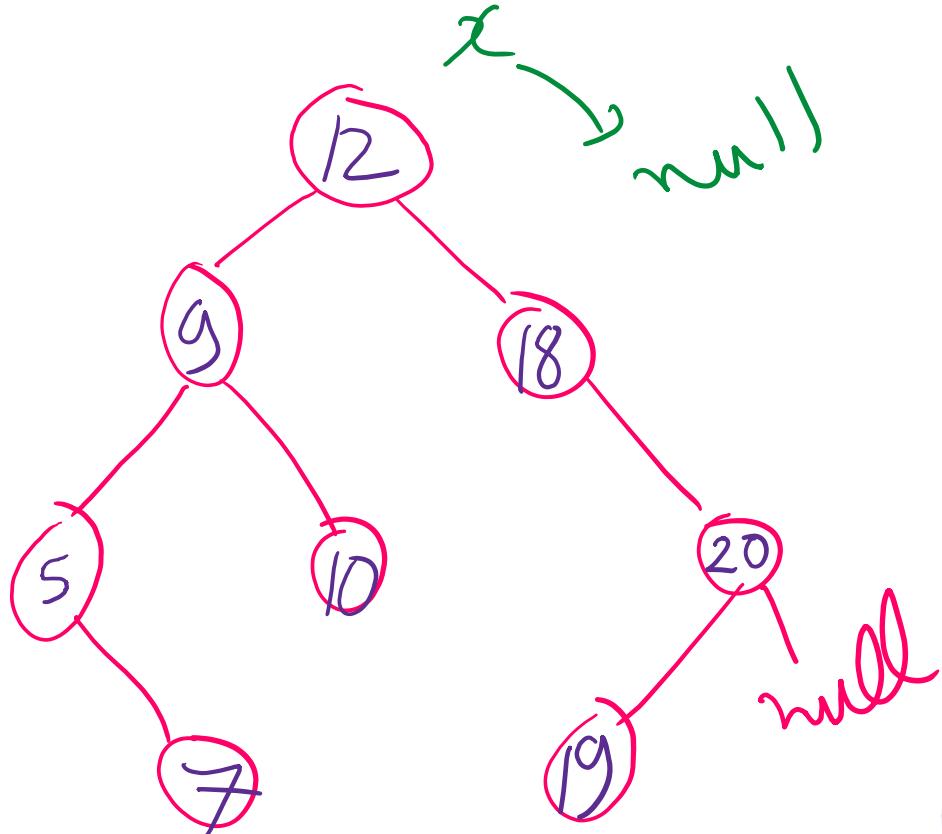
repeat until stack empty
& $x = \text{null}$

$x = \text{leftmost leaf}$
push to stack while
moving down
visit x

if x is right child
pop & visit
skip leftmost
leaf

Visit order: 7, 5, 10, 9, 19, 20, 18, 12
else $x = \text{parent. right}$

Iterative Postorder traversal



Stack
(explicit)

repeat until stack empty
& $x = \text{null}$

$x = \text{leftmost leaf}$
push to stack while
moving down
visit x

if x is right child
pop & visit
skip leftmost
leaf

Visit order: 7, 5, 10, 9, 19, 20, 18, 12
else $x = \text{parent. right}$

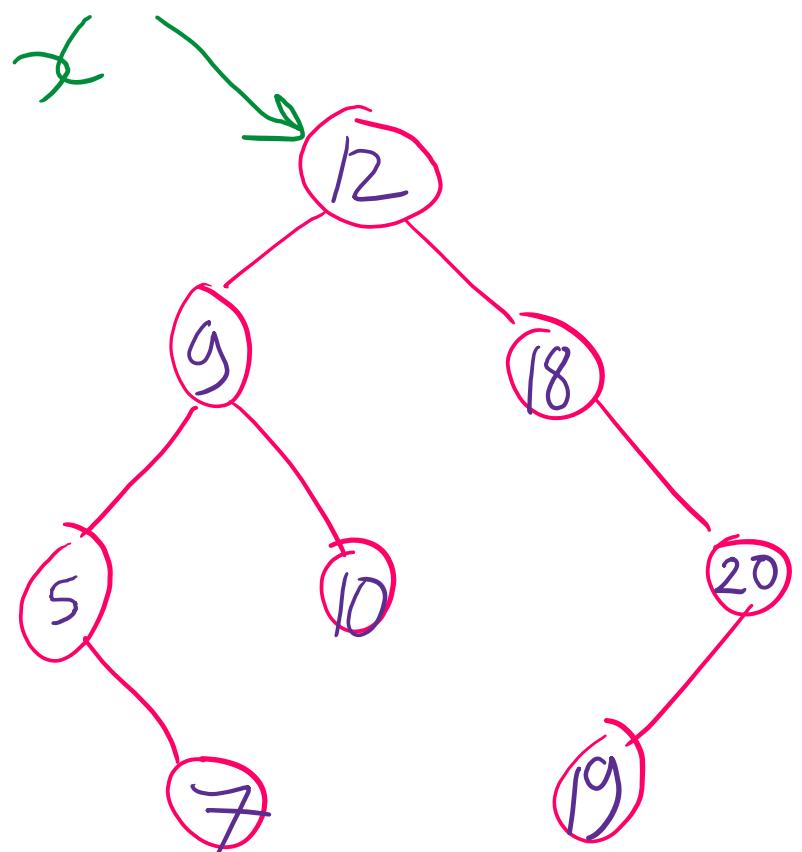
Traversals of a Binary Tree

- Preorder traversal
 - Visit root before we visit root's subtrees
- Inorder traversal
 - Visit root of a binary tree between visiting nodes in root's subtrees.
- Postorder traversal
 - Visit root of a binary tree after visiting nodes in root's subtrees
- Level-order traversal
 - Begin at root and visit nodes one level at a time
 - We will see the implementation when we learn Breadth-First Search of Graphs

Traversals of Binary Search Trees

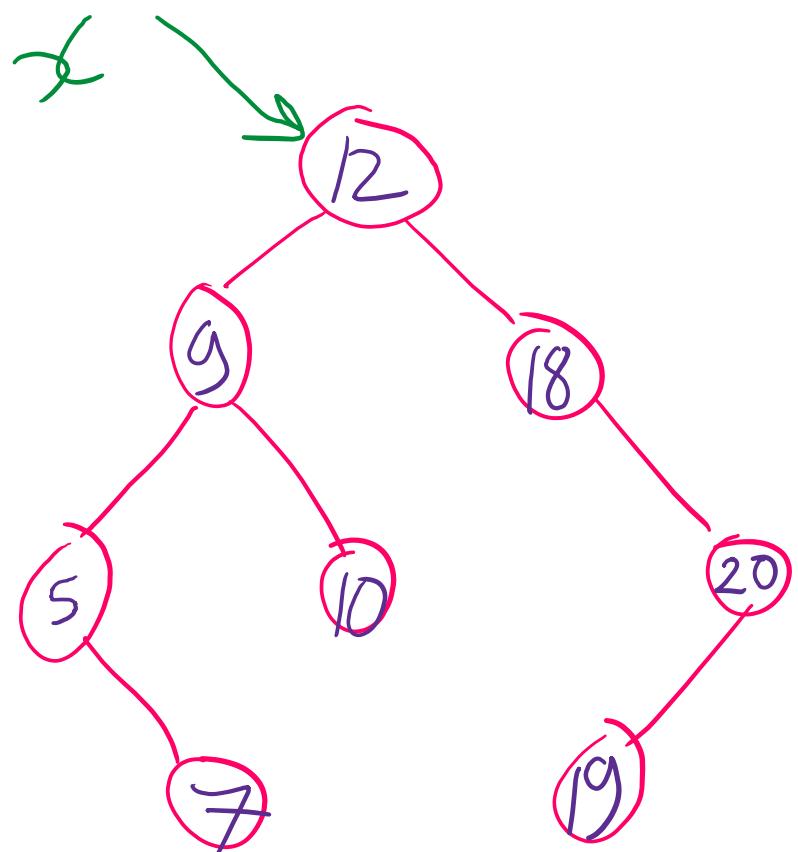
- Traversal for enumerating all items
- Traversal for finding an item

BST search using iteration



$x = \text{root}$
while($x \neq \text{null}$)
if equal break;
if $<$ $x = x.\text{left}$
if $>$ $x = x.\text{right}$
}

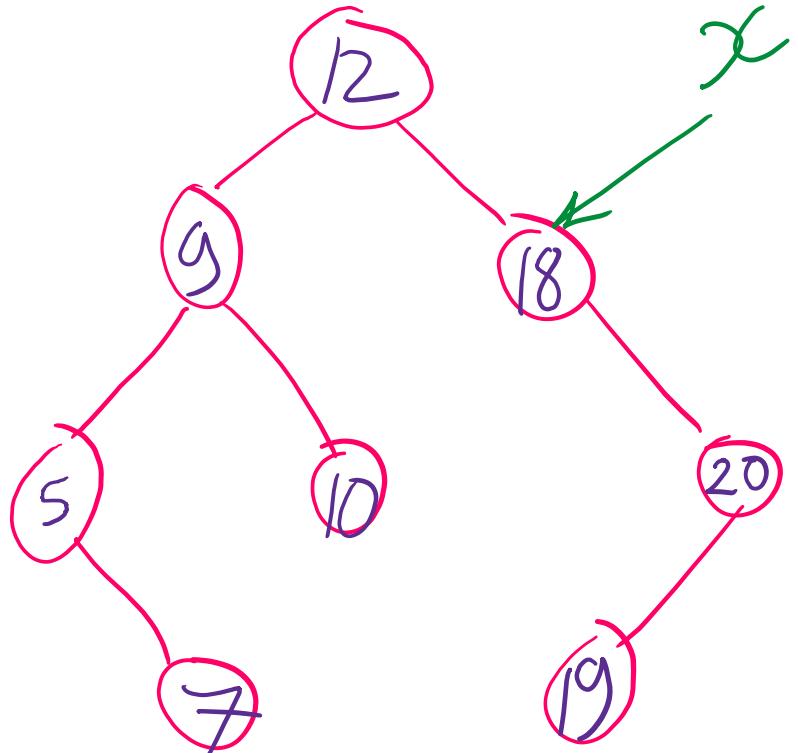
BST search using iteration



```
x = root  
while(x != null){  
    if equal break;  
    if < x=x.left  
    if > x=x.right  
}
```

Search for 19

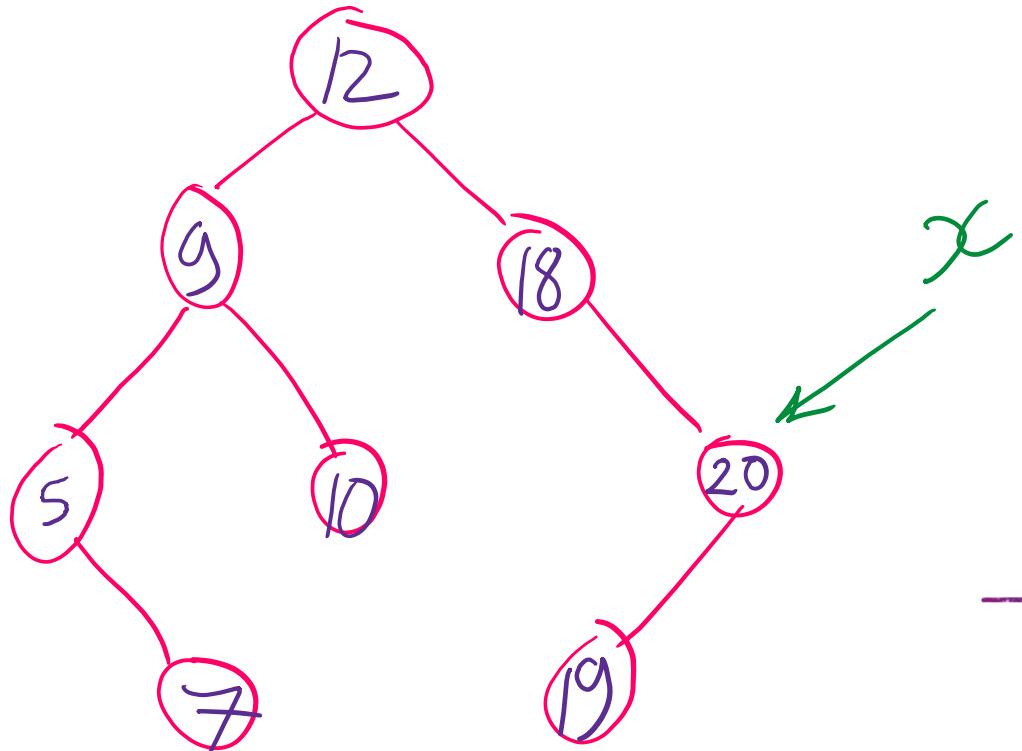
BST search using iteration



```
x = root  
while(x != null){  
    if equal break;  
    if < x = x.left  
    if > x = x.right  
}
```

Search for 19

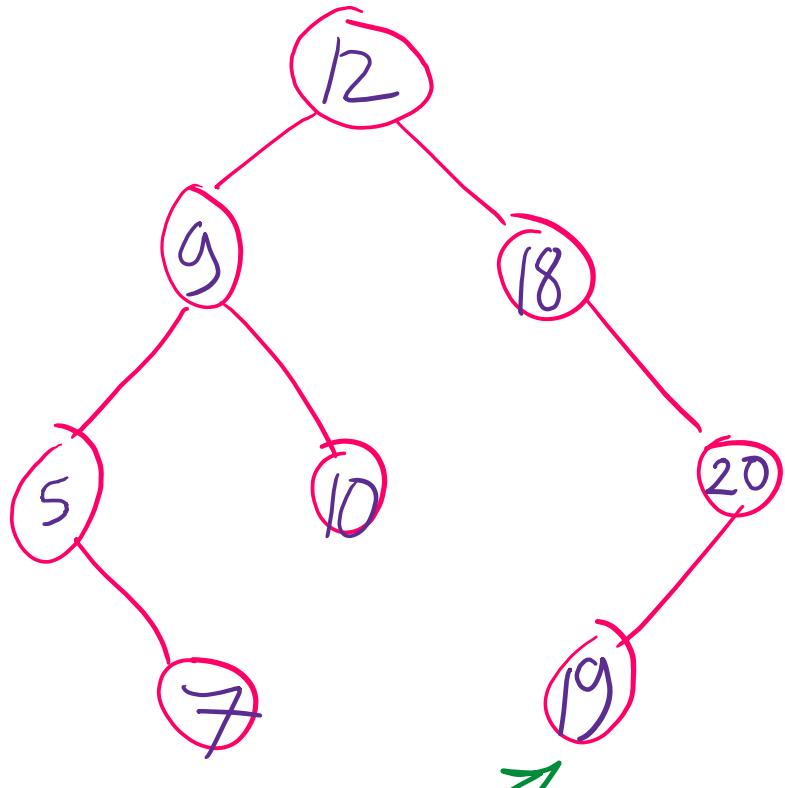
BST search using iteration



```
x = root  
while(x != null){  
    if equal break;  
    if < x = x.left  
    if > x = x.right  
}
```

Search for 19

BST search using iteration

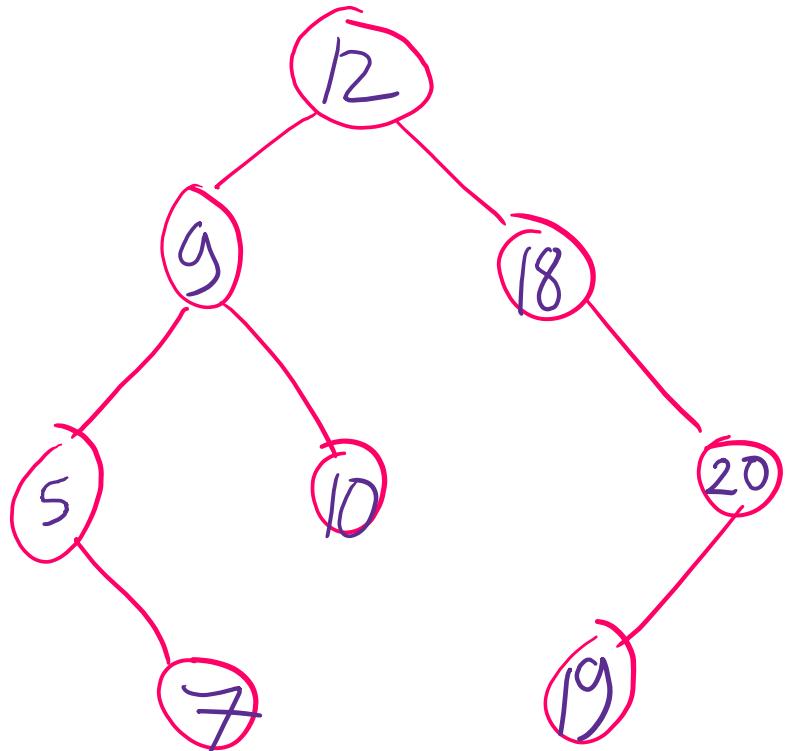


x

Search for 19

```
 $x = \text{root}$ 
while( $x \neq \text{null}$ ){
    if equal break;
    if  $<$   $x = x.\text{left}$ 
    if  $>$   $x = x.\text{right}$ 
}
```

BST search using iteration



x Search for 19

```
 $x = \text{root}$ 
\text{while}(x \neq \text{null})\{
    \text{if equal break;}
    \text{if } < x = x.\text{left}
    \text{if } > x = x.\text{right}
}
```